Hardward and algorithm architectures for real-time additive synthesis

Thesis

UNRESTRICTED

Hardware and Algorithm Architectures for Real-Time Additive Synthesis

A thesis submitted to the Open University

in partial fulfilment of the requirements for

the degree of

Doctor of Philosophy.

By

Peter Robert Symons

BSc (Eng) Hons, BSc (Open) Hons

Department of Information and Communication Technologies

of the Open University

June 2005

DATE OF SUBMISSION 29 MARCH 2005

DATE OF AWARD 29 JUNE 2005

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Abstract

Additive synthesis is a fundamental computer music synthesis paradigm tracing its origins to the work of Fourier and Helmholtz. Rudimentary implementation linearly combines harmonic sinusoids (or *partials*) to generate tones whose perceived timbral characteristics are a strong function of the partial amplitude spectrum. Having evolved over time, additive synthesis describes a collection of algorithms each characterised by the time-varying linear combination of basis components to generate temporal evolution of timbre. Basis components include exactly harmonic partials, inharmonic partials with time-varying frequency or non-sinusoidal waveforms each with distinct spectral characteristics. Additive synthesis of polyphonic musical instrument tones requires a large number of independently controlled partials incurring a large computational overhead whose investigation and reduction is a key motivator for this work.

The thesis begins with a review of prevalent synthesis techniques setting additive synthesis in context and introducing the *spectrum modelling* paradigm which provides baseline spectral data to the additive synthesis process obtained from the analysis of natural sounds. We proceed to investigate recursive and phase accumulating digital sinusoidal oscillator algorithms, defining specific metrics to quantify relative performance. The concepts of *phase accumulation*, *table lookup phase-amplitude mapping* and *interpolated fractional addressing* are introduced and developed and shown to underpin an additive synthesis subclass – *wavetable lookup synthesis* (WLS). WLS performance is simulated against specific metrics and parameter conditions peculiar to computer music requirements. We conclude by presenting processing architectures which accelerate computational throughput of specific WLS operations and the sinusoidal additive synthesis model. In particular, we introduce and investigate the concept of *phase domain processing* and present several "pipeline friendly" arithmetic architectures using this technique which implement the additive synthesis of sinusoidal partials.

Acknowledgements

I would like to thank Dr Mike Meade and Dr Terry McCarthy of the Open University Department of Information and Communication Technologies for their support, guidance and constructive input throughout the duration of this project. I would particularly like to thank Karen, my ever-loving wife and three children Chris, Ben and Jenny, without whose support, tolerance and love this project would never have concluded. Thanks are also due to my old friend Mike McNabb for typographical advice during the formatting of this thesis.

I dedicate this work to Karen and my parents Kath and Bob Symons.

Contents

List of Figures

(Figure captions have been précised for brevity.)

List of Tables

(Table captions have been précised for brevity.)

Glossary of Definitions and Common Terms

$x \in [a,b]$	$a \le x \le b$
$x \in [a,b)$	$a \le x < b$
$x \in (a,b]$	$a < x \le b$
$x \in (a,b)$	$a < x < b$
$\lfloor x \rfloor$	The integer part of x (i.e. the largest integer $\le x$).
$\lceil x \rceil$	The smallest integer $\ge x$.
$\langle x \rangle_y$	x modulo y and defined by $\langle x \rangle_y = x - y\left\lfloor \frac{x}{y} \right\rfloor$.
$\gcd(x, y)$	Greatest common divisor of x and y.
$\wedge$	Boolean AND operator.
$\vee$	Boolean OR operator.
$A_j(n)$	j^{th} time-varying amplitude sequence at time-index n.
$\alpha(n)$	Phase fraction sequence at time-index n.
B	Base frequency within an equally tempered tuning model.
$\beta_j(n)$	j^{th} partial pitch or frequency control parameter.
$\beta_j(\alpha)$	j^{th} interpolation coefficient function with argument α.
$\mathbf{C}_i[a]$	i^{th} interpolation coefficient lookup table vector with address a.
F	Fraction field width in bits after truncation by R bits.
$\hat{F}$	Total available fraction field width in bits equivalent to $M - I$.
$\mathbf{F}[a]$	First-order difference vector at address a.
f_s	Sample rate in Hz.

I	Integer field width in bits.
k_j	j^{th} harmonic integer multiplier.
L	Wavetable length in samples.
M	Phase accumulator word size in bits.
N	Interpolation order.
N_s	Number of samples computed in a particular analysis vector.
N_p	Number of partials within an additive synthesis model.
N_h	Number of harmonics within an additive synthesis model.
N_c	Number of computation time-slots within an arithmetic model.
p_a	a^{th} consecutive access vector memory sample-type index.
$\Re$	The set of all real numbers.
R	Number of bits truncated from an $\hat{F}$-bit fraction field.
r_1	First roll-off slope parameter for piecewise-linear envelopes.
r_2	Second roll-off slope parameter for piecewise-linear envelopes.
T	Sample period in seconds.
$\phi(n)$	Phase sequence at time-index n.
$\phi_I(n)$	Integer field of a phase sequence at time-index n.
$\phi_{Ir}(n)$	Rounded integer field of a phase sequence at time-index n.
$\phi_F(n)$	Fraction field of a phase sequence at time-index n.
$\phi_j(n)$	j^{th} time-varying phase sequence at time-index n.
$\phi_a(n)$	a^{th} consecutive access vector memory block address (denoted by ϕ_a for brevity).
$\Phi_j(n)$	j^{th} time-varying phase offset sequence at time-index n.

Φ_j	j^{th} start phase parameter.
ρ	Pitch control parameter.
I_ρ	Integer field width of pitch control parameter in bits.
F_ρ	Fraction field width of pitch control parameter in bits.
γ	Minimum equally tempered tuning frequency ratio.
$\varphi(n)$	Phase increment sequence at time-index n.
$\mathbf{Y}[x]$	Value of the vector $\mathbf{Y}$ at location x.
$y(n)$	n^{th} value of the discrete-time sequence y.
$y_r(n)$	Reference sequence at time-index n.
$y(-x)$	Initial condition at discrete-time time index $-x$.
Z	The set of all integers.
$\{a,b,c,d\ldots\}$	The set of elements $a,b,c,d\ldots$.

Glossary of Acronyms

ADC	Analogue to digital converter
AS	Additive synthesis
CAVM	Consecutive access vector memory
CIPM	Cubic interpolation phase mapping
CORDIC	Coordinate rotation digital computer
CPU	Central processing unit
CT	Continuous time
DAC	Digital to analogue converter
DFT	Discrete Fourier transform
DMA	Direct memory access
DPM	Dual-port memory
DSP	Digital signal processing
DT	Discrete time
ENBW	Equivalent noise bandwidth
FFT	Fast Fourier transform
FFT^{-1}	Inverse fast Fourier transform
FIR	Finite impulse response
FM	Frequency modulation
FPGA	Field programmable gate array
GAS	Group additive synthesis
HAS	Harmonic additive synthesis
IC	Initial condition
IDFT	Inverse discrete Fourier transform

IFFT	Inverse fast Fourier transform
IIR	Infinite impulse response
JND	Just noticeable difference
LIPM	Linear interpolation phase mapping
LSB	Least significant bit
LTAS	Long term average spectrum
LUT	Lookup table
MSB	Most significant bit
MWS	Multiple wavetable synthesis
PAS	Partial additive synthesis
PCA	Principal components analysis
PDF	Probability density function
PWL	Piece-wise linear
QIPM	Quadrature interpolation phase mapping
RAM	Random access memory
RMS	Root mean square
RMS	Root mean square
ROM	Read only memory
RPM	Rounded phase mapping
SFDR	Spurious free dynamic range
SIS	Spectral interpolation synthesis
SMS	Spectral modelling synthesis
SNR	Signal to noise ratio
SQNR	Signal to quantisation noise ratio
STFT	Short-time Fourier transform

STS	Short time spectrum
TIPM	Trigonometric identity phase mapping
TPM	Truncated phase mapping
TSA	Time slot address
VCO	Voltage controlled oscillator
VLSI	Very large scale integration
WLS	Wavetable lookup synthesis

Publications Related to this Research

"*DDFS phase mapping technique.*" IEE Electronics Letters, 10th October 2002, Vol. 38, No. 21, pp. 1291-1292.

"*Phase-Continuous Frequency Change in the Direct-Form, Second Order Recursive Oscillator.*" Computer Music Journal, Volume 28, Issue 4, Winter 2004, pp. 40-48.

Chapter 1 Introduction

1.1 Background

The research reported in this thesis investigates algorithms and arithmetic processing models which facilitate real-time synthesis of computer music audio signals. The results of this work are also pertinent to signal processing problems which require precision frequency synthesis of signals with time-varying spectra. The Fourier synthesis model which underpins this work, represents a complex signal by a linear combination of sinusoidal basis components with harmonic or inharmonic frequency distribution. Conceptually, this model is extensible to include time-varying linear combination of non-sinusoidal[1] basis components and so we use *additive synthesis* to describe the underlying synthesis model in line with the computer music literature.

The essence of additive synthesis is that complex musical sounds, or *timbres*[2], may be generated by linearly combining manifold basis components each having distinct, time-varying amplitude and frequency. The perceptual contribution of an *individual* basis component is relatively small, but collectively they combine to define the *timbral evolution*[3] of a sound. The synthesis process begins with the extraction of basis weightings (and in some cases basis *functions*) from the analysis of a natural sound using the short-time Fourier transform (STFT[4]). The STFT maps an analysed signal into a two-dimensional time-frequency space enabling extraction of basis amplitude and frequency time-profiles or *envelopes* which are subsequently modified to realise new sounds through additive resynthesis.

---

[1] We use *non-sinusoidal* as a generalisation of any complex signal composed of linearly combined sinusoids with harmonic or inharmonic frequency distribution.
[2] Timbre describes the tonal quality or "colour" of a synthesised sound.
[3] Timbral evolution describes how the timbre of a synthesised sound changes over time.
[4] The STFT decomposes the signal to be analysed into overlapping segments bounded in time by a window function.

1.2 An Historical Overview of Computer Music and Additive Synthesis

Max Mathews in his seminal *Science* article, envisioned "*the digital computer as a musical instrument*" where sound is synthesised from a numerical description – conceptually "*sound from numbers*". Since the bandwidth and dynamic range of human auditory perception are bounded, Mathews reasoned that "*any perceivable sound can be so produced*" [Mathews, 1963]. In *The Technology of Computer Music* [Mathews, 1969], Mathews proposed what he considered as two fundamental problems with sound synthesis using a digital computer:

- the enormous amount of data needed to specify the "pressure function" which represents the ultimate abstraction of a particular sound and implies a very fast computer program to resynthesise this function in real-time;
- the requirement for a powerful programming language which provides an intuitive environment in which complex sound sequences can be coded according to defined syntactic rules.

The first problem is being steadily abated by the geometric progression in "performance[5]-to-cost" ratio of digital computer building blocks, notably: programmable logic arrays, digital signal processors, microprocessors and semiconductor memory. Furthermore, this progression shows no sign of abating at the present time. The second problem is presently unresolved, although a fundamental observation is evident: sound samples must be computed algorithmically from a "numerical specification" since it is both perceptually non-intuitive and logistically infeasible to enter them individually from scratch – how would the user know where to start? If we assume algorithmic sample computation, a further observation becomes

[5] "Performance" takes on many interpretations in this context: "instructions per second", benchmark execution time, "connectivity", capacity and access time are typical examples.

apparent: a large number of sound samples are generated from a much smaller set of "specification numbers" which we generalise as the *synthesis parameters* corresponding to a particular *synthesis algorithm*. Smith [1991] observes a fundamental difficulty with algorithmic digital synthesis as finding the smallest set of synthesis algorithms that span "the gamut of musically desirable sounds". A fundamental objective of computer music research is to find a *single* synthesis technique that spans the universe of musically desirable sounds and has an "intuitively predictable" relationship between the control parameters and the synthesised sound. Indeed, Smith [1991] observes:

"*It is helpful when a* [synthesis] *technique is intuitively predictable. Predictability is good, for example, when analogies exist with well-known musical instruments, familiar sounds from daily experience, or established forms of communication (speech sounds).*"

We therefore seek a synthesis algorithm having intuitive control parameterisation which is capable of synthesising a broad range of musically desirable sounds and is computationally feasible in real-time with suitably fast hardware.

The Music III programming language introduced the *unit-generator* concept for sound synthesis which was developed and extended in the later Music IV and Music V languages [Mathews, 1969]. A unit-generator represents a fundamental building block which executes elemental functions within more complex sound synthesis algorithms that are specified using the vocabulary and syntax of the particular language. A unit-generator accepts both control and audio parameters depending on function and produces a corresponding output signal. The Music V unit-generators included an oscillator, filter, adder, multiplier, random number generator and envelope generator which were similar in function to the voltage controlled oscillators, filters and amplifiers used in analogue synthesisers of the time [Moog, 1965]. This complement of

elemental synthesis functions enabled research into various synthesis forms reported in Roads [1996] and summarised in Smith [1991]. However, Music V synthesis environments were constrained by being non real-time and "processor hungry". It was not uncommon for researchers to spend hundreds of seconds of mainframe CPU time to produce just one second of synthesised sound samples which were stored in a peripheral buffer and fed to a digital to analogue converter to allow them to be heard. This was anything but real-time and did not encourage creativity. However, Music V and its descendants, such as the much enhanced Mus10, helped promulgate computer music research in the 1970s and motivated the development of specialised processing hardware to speed up the computation process to the point where the results of a particular algorithmic synthesis technique could be heard in real-time.

A cornerstone synthesis technique explored using the Music V environment was *additive synthesis*. Additive synthesis is founded on the mathematical technique of Fourier analysis and uses the linear combination of sinusoidal basis components whose baseline weightings are obtained from the analysis of a natural sound to synthesise new sounds by appropriate modification. The first reported analysis-driven additive synthesis of sound appears to be Jean-Claude Risset's analysis and resynthesis of trumpet tones using Music V in 1964 [Risset, 1985]. Additive synthesis provides generality and accessibility to the lowest levels of a sound's timbral composition.

The underlying concept of additive synthesis is centuries old, first applied in pipe organs through their multiple *register-stops*. By appropriate register-stop settings, the sounds of several pipes are combined for each key depressed on the organ keyboard, greatly enriching the overall sound [Roads, 1996]. The arrival of Fourier analysis originating from the work of Jean Baptiste Fourier on heat conduction in 1822, introduced the concept of spectral analysis of sound. Helmholtz [1863], was the first

person to describe musical timbre in terms of the spectral components of a sound. Helmholtz constructed a rudimentary additive synthesiser which comprised ten electrically excited tuning forks each feeding a matching Helmholtz resonator via a mechanical shutter to control amplitude. Varying the individual shutter settings produced different timbres and was probably the first additive synthesiser based on the concept of a Fourier series. In the early twentieth century, (circa 1901) Thaddeus Cahill's massive *Teleharmonium* summed the weighted outputs of numerous rotating electrical tone generators to create complex musical sound textures transmitted directly to subscriber's households via the telephone system [Roads, 1996]. More recently, Laurens Hammond developed the *tonewheel*, a miniature version of Cahill's Teleharmonium tone generator and incorporated it in the legendary Hammond organ which is a pure additive synthesis instrument. The Compton Electrone organ used the rotation of a disk in close proximity to a fixed plate to produce a periodically varying capacitance which in turn generates elemental tones that are combined in an additive synthesis fashion [Comerford, 1993].

The additive synthesis concept has been widely adopted by computer music researchers because of its rigorous mathematical foundations and generality, albeit with a high computational cost associated with the synthesis and combination of numerous basis components. It now forms the foundation of the *spectral modelling* paradigm, which provides an intuitive sound synthesis methodology from a *frequency domain* perspective in line with the auditory timbral perception model.

Historically, implementation of additive synthesis had been confined to research environments computing sound sequences in non real-time using mainframe computers. However, in October 1977 the Centre for Computer Research in Music and Acoustics (CCRMA) at Stanford University took delivery of the Systems Concepts Digital

Synthesiser, or "Samson Box" as it became known, named after its designer Peter Samson. The Samson Box was a hardware implementation of all unit-generator elements from the Music V environment, including 256 waveform generators, 128 modifiers and a comprehensive interconnection subsystem [Loy, 1981; Smith, 1991]. The waveform generators supported both amplitude and frequency envelopes and the modifier functions could be reconfigured as second-order filter sections, random-number generators or amplitude-modulators. The Samson Box provided one of the first environments for real-time execution of additive synthesis and other algorithms. However, Smith [1991] reports that the Samson Box was not a panacea and required considerable effort in developing support software and debugging tools. Although the Samson Box did not provide the ideal foundation for a generalised synthesis research tool, it did point the way to what was possible with a dedicated "coprocessor" controlled by software executing "man-machine interface" functions. The end of the 1970s saw the introduction of two landmark systems spawned from research-oriented systems like the Samson Box: the New England Digital Synclavier and the Fairlight Computer Musical Instrument (CMI). The Synclavier was a modular, component based system that supported multi-voice additive synthesis and other algorithmic synthesis techniques. The Fairlight CMI possessed a similar modular architecture and supported both additive and sampling synthesis under comprehensive software control. These systems enjoyed huge commercial success despite price tags on the order of $100,000 for "fully loaded" systems and demonstrated the need for both a live performance instrument and research-oriented system where cost was secondary to performance. Real-time signal generation is handled by dedicated hardware optimised to a particular synthesis algorithm. A host micro-computer undertakes all control, user interface and performance management functions with the dedicated synthesis hardware integrated as a *coprocessor* function.

1.3 Research Motivation and Objectives

1.3.1 Introduction

Historically, the application of sinusoidal additive synthesis has been hindered by a significant computational imposition, particularly in "orchestral synthesis" applications where multiple independently controlled voices[6] are required. For example, synthesising a single 27.5 Hz (A0) complex musical tone at a 48 kHz sample rate requires around 872 partials[7] if the full Nyquist[8] bandwidth is used. If we proceed to assume that such tones are synthesised within a 100-voice polyphonic ensemble, which is typical for a demanding orchestral synthesis environment, then approximately 87,000 partials will be needed under peak conditions. Clearly, this figure represents an absolute upper bound for this level of polyphony. However, an average figure of 100 partials per voice as suggested by [Smith and Cook, 1992] requires 10,000 partials and their respective control parameters to be computed in real-time. The logarithmic frequency response of the human ear and the observation by Sandell [1994] that the spectral amplitude envelope of musical timbres progressively diminishes at frequencies above 5 kHz suggests that, in general, low frequency partials should be assigned higher priority over high frequency partials for inclusion in the synthesised partial group. Accordingly, the additive synthesis computation burden is reducible through "perceptual coding" of the *composite* partial spectral envelope by pruning high frequency partials against a timbral perception "coding" model. The development of coding models which reflect human perceptual characteristics is currently an active area of research and complements the

[6] A *voice* describes a group of partials sharing a common fundamental and collectively forming a distinct synthesised sound with unique timbral identity.
[7] A *partial* is a generalised form of harmonic, describing a *sinusoidal* basis component that does not necessarily have an exact harmonic relationship to the fundamental.
[8] The *Nyquist bandwidth* (or *Nyquist region*) refers to the available bandwidth within a sampled system equivalent to one half of the sampling frequency. Frequency components greater than one half of the sampling frequency are aliased or "folded over" into the Nyquist region [Orfanidis, 1996].

pursuit of effective sinusoidal additive synthesis processors [Marentakis and Jensen, 2002; Jensen, 1999].

Synthesising 10,000 linearly combined partials at a 48 kHz sample rate requires a single partial sample computation every 2 ns, suggesting a single thread pipelined processor clocked at 480 MHz. The availability of high density programmable logic arrays [Xilinx, 2004] and multi-port memory [Smith, 2000] optimised for very high speed digital signal processing now makes such pipelined processors a practical proposition at a cost comparable to a high-end workstation computer system. Moreover, the cost-to-speed ratio of this technology is continuing to fall at this point in history.

1.3.2 Motivation

Motivation for this work stems from the observation that additive synthesis as defined by the linear combination of *sinusoidal* basis components provides complete accessibility to the elemental parts of timbral composition, precisely in line with accepted models of timbral perception [Jensen, 1999]. Other reported forms of the additive synthesis model utilise *non-sinusoidal* basis components [Horner *et al*, 1993; Kleczowski, 1989] and so we base this work on a definition of additive synthesis as the linear combination of *sinusoidal or non-sinusoidal* basis components with time-varying frequency, amplitude and phase parameters. Accordingly, this research is built on two hypotheses that encapsulate the research problem and are investigated and developed in subsequent chapters:

1. Phase accumulating frequency synthesis with concurrent interpolated table lookup phase-amplitude mapping provides an extensible computational environment for implementing all facets of the additive synthesis paradigm.

2. The underlying properties of modular phase accumulation may be exploited to generate phase sequences with harmonic and inharmonic frequencies and hence corresponding sinusoidal partials through phase-amplitude mapping.

1.3.3 Objectives

The principal objectives of this research are to investigate sinusoidal and non-sinusoidal waveform synthesis algorithms within an additive synthesis context and in line with the above hypotheses; their relative performance against defined metrics and processor[9] architectures that enable effective execution of the underlying arithmetic processes in real-time. The research is organised into essentially six focal topics, summarised below, each with distinct objectives and which collectively define the framework of this thesis as summarised in section (1.4).

1. Review prevalent synthesis techniques reported in the literature, justifying additive synthesis as the focus of this research and introduce the reported forms of the additive synthesis paradigm.
2. Review recursive and phase-accumulating sinusoidal oscillator algorithms reported in the literature, justifying phase-accumulation as the preferred technique for subsequent investigation and development.
3. Review reported phase-amplitude mapping techniques, introduce the concept of a wavetable and sampling a tabulated signal.
4. Investigate generalised phase-accumulating frequency synthesis and its application to sinusoidal and non-sinusoidal waveform synthesis.

[9] In the context of this research, *processor* typically describes a pipelined concatenation of processor elements where each element is optimised to execute a particular atomic operation from an arithmetic partitioning of the underlying algorithm.

5. Investigate phase-amplitude mapping based on interpolated table lookup. Develop computer models which support a comparative assessment of phase-amplitude mapping techniques with different tabulated signals.
6. Investigate arithmetic processing architectures which improve execution speed of the additive synthesis model and develop the concept of processing in the phase domain.

Research methodology is driven from a critical review of reported material and focuses on areas where the efficiency, flexibility or understanding of additive synthesis computation can be improved. An analytical approach substantiated by computer modelling is used to argue the effectiveness and validity of the presented material rather than the construction and assessment of demonstrable hardware or software. It is intended that the feasibility and efficacy of hardware processors based on the techniques presented should be self evident from the respective thesis discussion.

1.3.4 Assumptions

The development of synthesis algorithms presented in this thesis is predicated on four assumptions that we accept *a priori* based on the wider literature and which underpin high throughput algorithmic processing architectures operating in real-time.

1. Devolution of algorithmic synthesis operations to a "performance-optimised" coprocessor as an adjunct to a general purpose host computer, increases throughput, flexibility of control and ease of migration to new host platforms as they become available [Roads, 1996; Samson, 1980; Alonso *et al*, 1977].
2. At this point in history, an optimised "hardwired" processor provides greater computational throughput compared to a "software-driven" processor by trading flexibility for speed [Pirsch, 1996].

3. Arithmetic partitioning implicit within the synthesis and linear combination of manifold basis components which defines the additive synthesis paradigm, is naturally compatible with a pipelined processing architecture where processing stages are *individually* optimised to execute *specific* elemental operations [Pirsch, 1996].
4. Table lookup operations are generally faster than algorithmically computing the tabulated data in real-time. Utility improves with increasing arithmetic complexity of the underlying algorithm, offset only by lookup table memory length and hence cost [Pirsch, 1996; Chamberlin, 1985].

1.4 Thesis Structure

Chapter 1 introduces this thesis and presents an historical overview of additive synthesis and some of the hardware platforms that have been applied. Thesis motivation, objectives and structure are also outlined.

Chapter 2 reviews and categorises popular synthesis techniques and their underlying processing models reported in the literature. Techniques are classified according to a cited taxonomy and assessed against defined criteria peculiar to computer music utility. Our objective is to set additive synthesis in context relative to other techniques, justify the additive synthesis focus of this thesis and to introduce pertinent mathematical foundations for its implementation.

Chapter 3 reviews discrete-time recursive and phase-accumulating sinusoidal oscillator algorithms reported in the literature and critically reviews them against pertinent computer music metrics. Section (3.2.6) presents original published work resulting from this research which is concerned with the determination of initial condition values which provide phase-continuous frequency transition in the second-order direct-form recursive oscillator. Phase accumulation and phase-amplitude mapping are introduced

and several implementation techniques introduced and discussed, including *wavetable lookup* and *phase interpolation* which underpin subsequent chapters.

Chapter 4 investigates frequency synthesis by phase-accumulation and wavetable lookup as the basis of computationally efficient phase-amplitude mapping. The chapter investigates frequency control resolution specific to computer music requirements and introduces the concepts of *phase truncation* and *fractional addressing* of a tabulated signal by interpolation.

Chapter 5 develops the fractional addressing concept and investigates interpolated[10] phase-amplitude mapping as a refinement of the wavetable lookup technique. Phase-amplitude mapping by trigonometric identity is introduced in section (5.3) and shown to provide optimal performance bound only by sample quantisation noise. This material represents original work resulting from this research which is now published. The chapter concludes by presenting simulated qualitative performance data for several interpolated phase-amplitude mapping techniques over a range of wavetable spectrum characteristics pertinent to musical application.

Chapter 6 investigates memory access associated with the interpolated phase-amplitude mapping model and presents an original memory architecture with concurrent interpolation processing which enables efficient data-parallel execution of this model for various interpolation orders. We briefly review the concept of block pipelining using dual-port memory which decouples parameter update and real-time algorithmic sample computation and underpins pipelined processor architectures presented later. Introducing the concept of *phase domain* processing which algorithmically modifies phase information to effect frequency control, we proceed to consider arithmetic models and corresponding hardware architectures that compute phase sequences and hence

sinusoids with harmonically related frequencies whose mathematical basis was first mooted at the end of Chapter 4. Finally, we extend this technique to include inharmonic sinusoids (i.e. true partials) and discuss the architecture of a pipelined sinusoidal additive synthesis coprocessor.

Chapter 7 summarises this thesis in light of the research objectives set out in Chapter 1. Limitations of the presented work and areas for further research and investigation are discussed.

Appendix A presents an introductory overview of Lagrange and Newton polynomial interpolation which supports discussion within Chapter 5. The concept of the interpolation sample set is introduced and optimal placing of this set with respect to the fractional address interval is discussed.

Appendix B presents documented Mathcad model listings which are applied in Chapter 5 to assess qualitative performance of interpolated phase-amplitude mapping algorithms and in Chapter 6 to demonstrate HAS and PAS process models presented therein.

Appendix C presents the order-3 consecutive access vector memory architecture including an arithmetic model for effecting the modulo division-by-3 needed within the block addressing computations.

1.5 Original Work

The following areas represent original results from this research:

- Definition of initial condition values for the second-order direct-form recursive oscillator which afford exact phase-continuous frequency transition with constant amplitude. The algorithm is shown to incur significant computation imposition and aside from generalising the utility of this recursive oscillator

[10] We define *interpolation* as the computation of sample values at non-tabulated points according to an implicit fractional table address.

which is of academic interest, serves to strengthen the case for phase-accumulating techniques.

- A sinusoidal phase-amplitude mapping technique is presented which provides interpolation error and therefore signal-to-noise ratio bound by sample quantisation noise alone. In essence, an M-bit phase word is optimally mapped to the amplitude domain using a "virtual" 2^M location sinusoidal lookup table requiring only $2^{\frac{M}{2}+2}$ memory locations. The technique permits sinusoid phase control with a resolution of $\frac{2\pi}{2^M}$ radians.
- A wavetable memory architecture which enhances computational efficiency of the interpolated table lookup processing model. The concept of a new wavetable memory architecture, the *consecutive access vector memory* (CAVM), is presented and which is extensible to improve interpolated indexing of any tabulated data space under certain conditions.
- Processing architectures for generating non-consecutive harmonic and inharmonic partial phase sequences which enables implementation of the harmonic and partial additive synthesis processing models.

Chapter 2 Computer Music Synthesis Techniques

2.1 Introduction

In this chapter we review the foremost discrete-time synthesis techniques reported in the literature, focussing on *additive synthesis* (AS) techniques and their respective signal processing models. Our aim is to identify and scope the focal areas that underpin later chapters in this thesis and which support investigation of processor architectures computing particular forms of the AS paradigm in real-time. We define a *synthesis technique* as an algorithm that processes parametric control information to produce "musically useful" sound samples, executing in software or hardware. We structure our review using the taxonomy of synthesis techniques proposed by Smith [1991] which comprises – *processed recording, spectral modelling, physical modelling* and *abstract algorithm*. We consider the ten assessment criteria proposed by Jaffe [1995]:

1. *Intuitiveness* of control parameters
2. *Perceptibility* of a control parameter *change*
3. *Physicality* of control parameters
4. Control parameter *behaviour*
5. Robustness of the synthesised sound's *identity*
6. *Classes* of sound represented
7. Synthesis algorithm *efficiency*
8. Synthesis algorithm *latency*
9. Control stream *sparseness*
10. Existence of corresponding *analysis tools*

An intuitive control parameter articulates a musically expressive attribute such as timbre in a perceptually meaningful way. The perceptibility of a control parameter change assesses the audible effect corresponding to that change and classifies the association from strong to weak. The physicality of a parameter describes how well that parameter controls a synthesised instrument in an analogous manner to its natural counterpart.

The behaviour of a control parameter considers the proportionality between a parameter change and the corresponding perceived auditory change. A small parameter change which produces a large auditory change is undesirable and reflects a parameter which is not well behaved, possibly difficult to control and non-intuitive. Maintaining the identity of a sound following parametric change reflects the robustness of the corresponding synthesis technique (e.g. does a violin still *sound* like a violin following a parameter change – albeit a *different* violin?).

Sound classification considers the range of sounds possible with a particular synthesis technique and more importantly – what classes of sound are *not* possible? Algorithm efficiency considers the memory, computational power and control data bandwidth required in the execution of a synthesis technique and dictates the number of voices that can be synthesised with a given processing capacity and sample rate. Algorithm latency is a critical consideration in real-time synthesis and describes the delay between a parametric change and the corresponding audible effect. The sparseness of the control parameter stream assesses the control processing overhead against the arithmetic complexity of the synthesis algorithm (i.e. where is the work being done?). The availability of sound analysis techniques which generate baseline synthesis parameters matched to a particular synthesis technique is crucial. Indeed, Jaffe [1995] observes:

"*It is not enough to know in theory that any sound can be produced. You need tools to derive the proper parameter values from a specification of a desired result.*"

The objective is to commence the synthesis process from an "identity baseline" which reproduces a natural instrument sound using the particular synthesis technique and then modify synthesis parameters to generate new sounds which are derived from the original baseline sound.

2.2 Processed Recording

2.2.1 Sampling Synthesis

Processed recording describes synthesis techniques based on *time-domain* transformation of pre-recorded or pre-computed sounds. Sampling synthesis is the most prevalent technique reported, and involves "pitch shifting" a pre-recorded sound relative to the original pitch using sample rate conversion followed by time-varying filtration to modify timbral evolution and articulation. Many instruments exhibit perceptually important transient behaviour at the onset of a note (i.e. the attack phase) which must be captured if the resynthesised sound is to be recognisable. Winckel [1967] reports that musical instruments are identified principally by their attack characteristics. Playback sustain (i.e. the continuation of a sound following the attack phase) is accomplished by "looping" selected segments of the recording via appropriate memory addressing. All of these operations add perceptible distortions to the resynthesised sound and detract from the objective of imitating and modifying pre-recorded instruments. These distortions can be mitigated by *multi-sampling* an instrument sound across sub-intervals of its pitch range and forming a library of separate recordings (so called *samples*) which are indexed on playback as a function of the resynthesised pitch parameter [Massie, 1998].

Smith [1991] reports an inherent and fundamental problem with sampling synthesis as its lack of "prosodic rules" for musical phrasing upon playback. Resynthesised notes

played individually are realistic reproductions of the original, but notes played in sequence lack the note-to-note transitions which characterise many real instruments.

Control parameterisations may be applied to pitch shifting, sustain looping and post-synthesis filtration operations, providing a somewhat limited and non-intuitive "control space" to support the creation of new timbres. Pitch shifting methods based on time-domain transformation produce "temporal distortion" of the original sound (i.e. compression or stretching of salient temporal features) causing most sounds to lose their timbral identity after only a few semitones pitch shift relative to the original [Jaffe, 1995; Smith, 1991].

Timbral control and articulation is effected by fundamentally two means:

1. Using multiple acquisitions (a.k.a. "samples") of the underlying sound captured over pertinent parametric sub-intervals (e.g. pitch or key depression velocity).
2. Application of time or parametrically varying filtering to the resynthesised sound.

Implementation efficiency is a strong function of the pitch shifting interpolation algorithm which typically requires multiple accesses to the sample memory [Massie, 1998].

2.2.2 Wavetable Lookup Synthesis

Wavetable lookup synthesis (WLS) is a classic technique where a single period of an arbitrary waveform is tabulated in memory and cyclically addressed to resynthesise a periodic sound [Roads, 1996]. An extension of the multi-sampling technique applied in sampling synthesis assembles a contiguous wavetable sequence where each wavetable contains a single-cycle "snapshot" waveform across an instrument's "timbre space". The wavetable set is read consecutively to resynthesise the sound, providing significant

data reduction compared to the original sample. Smoother timbral articulation is obtained by time-domain interpolation between adjacent wavetable timbres – so-called *spectral interpolation synthesis* (SIS) [Serra *et al*, 1990].

WLS is not confined to a uni-dimensional "waveform-time" representation. Multi-dimensional forms are reported comprising wavetables which snapshot an instrument's timbre over a corresponding multi-dimensional parameter space. Assuming mutually independent parameters (e.g. time, pitch and key depression velocity), each dimension snapshots timbre on a particular parameter interval with all other parameter values fixed. Assuming timbre change over a particular parameter interval (i.e. dimension) is well behaved, a coarse timbral quantisation on the corresponding dimension is permissible with linear interpolation used to compute intermediate timbres from those tabulated in a similar manner to the SIS model. The degree of permissible quantisation is governed by perceptual constraints. Multi-dimensional WLS is generally known as *vector synthesis* in the literature [Smith, 1991]. Timbre is now represented as a vector quantity computed by linear interpolation of an n-dimensional timbre-space according to n parameters and requiring 2^n interpolation points [Wessel, 1979; Haken, 1991]. However, vector synthesis incurs exponentially increasing memory size and interpolation processing overhead as n grows.

A development of WLS – *multiple wavetable synthesis* (MWS), is illustrated in Figure (2.2.1) and based on time-varying linear combination (i.e. weighted summation) of fixed, periodic basis functions stored as wavetables [Horner *et al*, 1993]. The individual wavetables are indexed with the same time-varying phase function (whose slope represents frequency) and so all basis functions have identical phase and frequency.

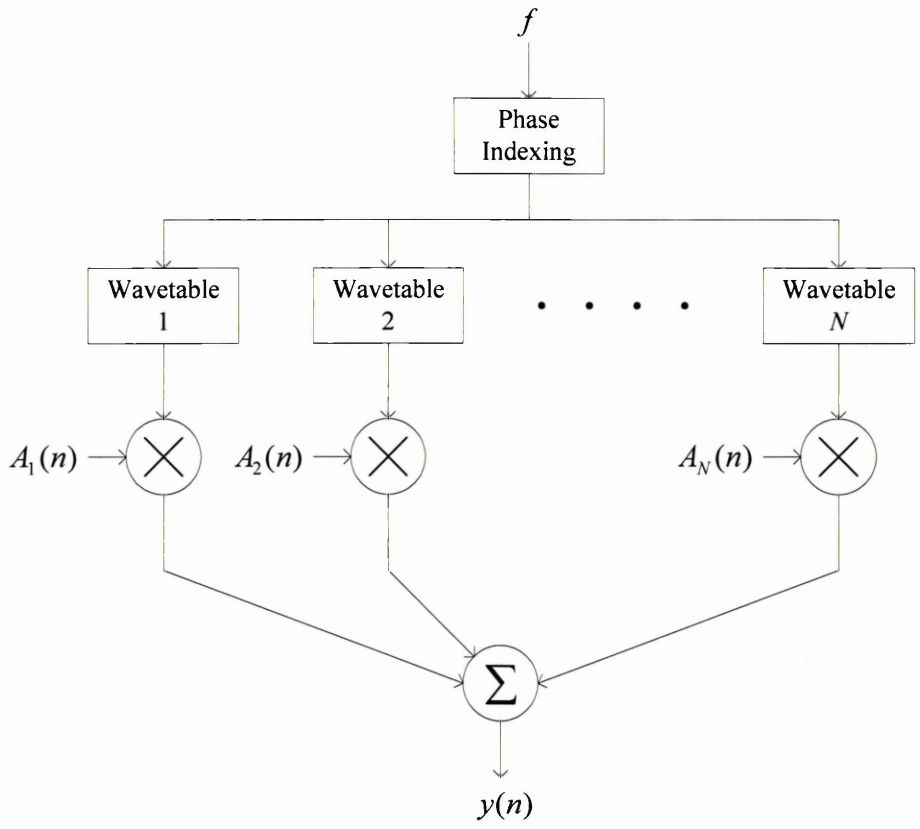


Figure (2.2.1): The multiple wavetable synthesis (MWS) algorithm. N distinct wavetables are linearly combined with time-varying weights $A_k(n)$, $k \in [1, N]$.

Setting the $A_k(n)$ weighting terms to overlapped triangular window functions interpolates the wavetable set contiguously and reduces MWS to the SIS model. If the wavetable harmonic sets comprise distinct harmonic groupings, Horner *et al* [1993] observe that MWS approaches *group additive synthesis* (GAS) where the complete set of partials making up a sampled sound are organised into several wavetables each having a common partial amplitude and frequency-time profile [Kleczkowski, 1989]. However, a key distinction is that GAS applies a unique time-varying amplitude *and* frequency weighting to each wavetable in the group, whereas MWS applies only time-varying amplitude weighting. The GAS processing model is illustrated in Figure (2.2.2) where we observe that each basis component has time-varying amplitude and frequency parameterisation.

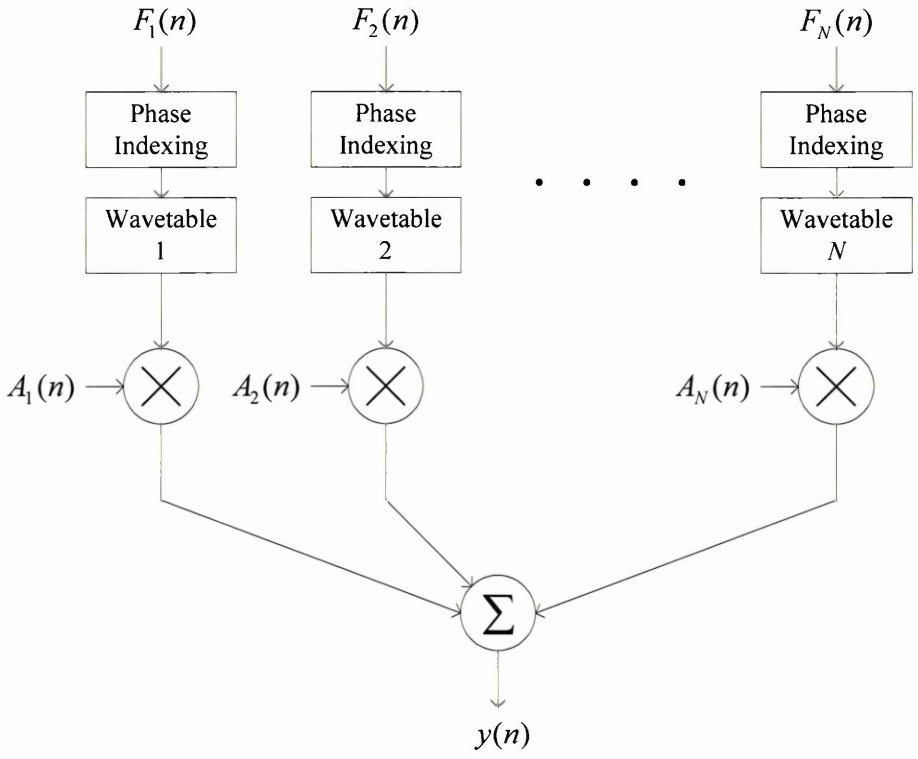


Figure (2.2.2): Generalisation of multiple wavetable synthesis (MWS) to group additive synthesis (GAS), where each wavetable has a unique time-varying amplitude and frequency weighting denoted by $A_k(n)$ and $F_k(n)$, respectively.

The determination of basis wavetables which support the MWS model is reported in Stapleton and Bass [1988] and Horner *et al* [1993]. MWS and its derivatives have been utilised successfully in several commercial music synthesisers, typically with wavetables containing natural sound samples. Several variants are reported in the literature, differentiated principally by the combination process model [Roads, 1996].

2.3 Spectrum Modelling

2.3.1 The Fourier Transform

Spectrum modelling describes the subclass of synthesis techniques which specify musical sounds in the frequency domain. Fourier's theorem states that any periodic function can be represented as a sum of harmonically related sinusoids each with a

particular amplitude and phase. A typical frequency domain representation therefore comprises a spectrum of discrete "lines" corresponding to sinusoidal basis functions whose baseline values are usually determined from the analysis of a natural instrument sound.

The discrete Fourier transform (DFT) underpins the spectrum modelling paradigm and provides bidirectional transformation between the discrete time and frequency domains as illustrated in Figure (2.3.1). For sequences of length N and sample period T, we have:

$$Y(k) = \sum_{n=0}^{N-1} y(nT) e^{-jk\Omega nT}$$

$$y(nT) = \frac{1}{N} \sum_{k=0}^{N-1} Y(k) e^{jk\Omega nT} \qquad (2.3.1)$$

$$k \in [0, N-1] \qquad n \in [0, N-1]$$

where $y(nT)$ and $Y(k)$ represent the respective time and frequency sequences. Ω denotes the first *harmonic frequency* given by $\Omega = \frac{2\pi}{(N-1)T} \approx \frac{2\pi}{NT}$ for $N >> 1$ and determines the frequency spacing of $Y(k)$. These transforms provide the basis for analysis, modification and subsequent resynthesis of musical signals (indeed any periodic signal) where the intermediate representation exists in the frequency domain as a weighted sum of sinusoidal basis functions.

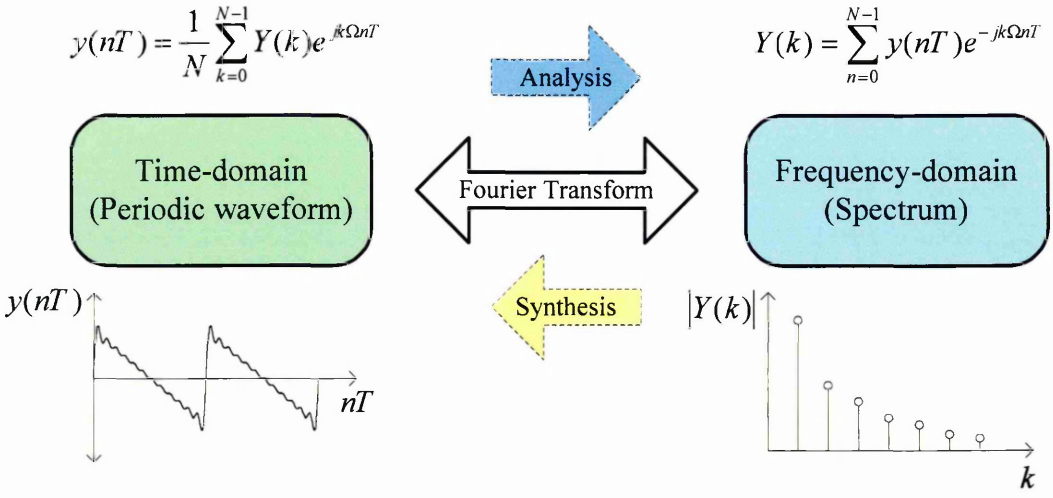


Figure (2.3.1): The discrete Fourier transform (DFT) pair.

The analysis or forward DFT (i.e. time to frequency domain transformation) generates a complex sequence, $Y(k)$, which represents the *magnitude spectrum*, $M(k) = |Y(k)|$, and *phase spectrum*, $\Phi(k) = \angle Y(k) = \tan^{-1}\left(\frac{\mathrm{Re}(Y(k))}{\mathrm{Im}(Y(k))}\right)$, corresponding to the complex DT sequence, $y(n)$, where k represents the discrete frequency index. The discrete spectrum is an *estimate* of the true spectrum bound by the transform frequency resolution given by $\frac{2\pi}{NT}$. The "bin frequency" corresponding to the k^{th} location of the discrete spectrum is given by $\frac{2\pi k}{NT}$ [Ifeachor and Jervis, 1993].

The synthesis or inverse DFT (i.e. frequency to time domain transformation) generates a complex DT sequence, $y(n)$, corresponding to the complex discrete spectrum, $Y(k)$.

The forward DFT is usually computed using the *fast Fourier transform* (FFT) algorithm which exploits computational redundancy inherent in the DFT to significantly reduce computation time [Rabiner and Gold, 1975]. It is reported in Ifeachor and Jervis [1993] that computational savings afforded by the FFT as compared to the DFT increase as

$N^2 - \left(\frac{N}{2}\right)\log_2 N$. It is evident that DT audio signals are represented by *real* sequences (i.e. have a zero imaginary part) which leads to superfluous computation in the complex FFT. Specific algorithms are reported which transform real DT sequences without redundant computation thereby improving the computation speed of audio spectrum analysis [Chamberlin, 1985].

2.3.2 Sinusoidal Additive Synthesis

We define sinusoidal additive synthesis as the time-varying linear combination of *sinusoidal* basis components. Two distinct forms are reported in the literature: *harmonic additive synthesis* (HAS) and *partial additive synthesis* (PAS), with PAS the more general of the two forms. HAS synthesises precisely periodic signals according to a *harmonic spectrum model* whose basis frequencies follow a harmonic distribution in line with the inverse DFT. PAS synthesises signals according to a *partial spectrum model* whose basis components or *partials* have time-varying frequency. We therefore define a *partial* as a generalised frequency component with time-varying frequency and a *harmonic* as the special case constrained to a frequency distribution which follows an exact harmonic series. Both HAS and PAS algorithms process time-varying parameter *envelopes* defined as the time-contour (or *trajectory*) described by a parameter relative to some initiation event (e.g. a key depression).

Before presenting discrete-time definitions of the PAS and HAS algorithms, we first consider the continuous-time phase variation, $\phi(t)$, corresponding to an instantaneous frequency, $f(t)$, given by:

$$\phi(t) = 2\pi \int_0^t f(u)du + \Phi(t) \qquad (2.3.2)$$

where $\Phi(t)$ represents a time-varying phase offset. We now proceed to express the *continuous-time* PAS model which synthesises a signal, $y(t)$, from N_p sinusoid partials. Each partial has amplitude, frequency and start phase envelopes respectively denoted by $A_k(t)$, $f_k(t)$ and $\Phi_k(t)$, hence we obtain:

$$y(t) = \sum_{k=1}^{N_p} A_k(t)\cos(\phi_k(t))$$
$$\phi_k(t) = 2\pi \int_0^t f_k(u)du + \Phi_k(t) \qquad (2.3.3)$$

where $\phi_k(t)$ represents the k^{th} partial continuous-time phase function. The corresponding HAS model is obtained by setting $f_k(t) = kf_0(t)$ in Eqs. (2.3.3), where $f_0(t)$ denotes the fundamental frequency envelope function. For constant fundamental frequency we have $f_k(t) = kf_0$.

In a discrete-time system, the instantaneous frequency is represented by a sequence, $F(n)$ and for a sample period, T, the corresponding phase sequence is given by the discrete-time equivalent of Eq. (2.3.2), thus:

$$\phi(n) = 2\pi T \sum_{m=1}^{n} F(m) + \Phi(n) \qquad (2.3.4)$$

where m is analogous to the dummy variable u in Eq. (2.3.2). If the instantaneous frequency is constant (i.e. $f(t) = f'$ for all t and $F(n) = F'$ for all n) we have $\phi(t) = 2\pi f' t + \Phi(t)$ and $\phi(n) = 2\pi F' nT + \Phi(n)$ for the CT and DT cases, respectively. The inverse DFT may be expressed in a real form which computes the sequence $y(n)$ from a linear combination of N_p *partials* and therefore defines the PAS model, thus:

$$y(n) = \sum_{k=1}^{N_p} A_k(n)\cos(\phi_k(n))$$

$$\phi_k(n) = 2\pi T \sum_{m=1}^{n} F_k(m) + \Phi_k(n) \tag{2.3.5}$$

where the k^{th} partial phase sequence is denoted by $\phi_k(n)$ with $A_k(n)$, $F_k(n)$ and $\Phi_k(n)$ respectively denoting the k^{th} partial amplitude, frequency and phase envelopes that collectively embody the spectrum model. The frequency envelope, $F_k(n)$, is peculiar to PAS where some partial frequencies are time-variant reflecting the behaviour of natural instruments [De Poli, 1983].

In an analogous manner to the continuous-time case, the corresponding HAS model is obtained by setting $F_k(n) = kF_0(n)$ in Eqs. (2.3.5), where $F_0(n)$ denotes the *fundamental* discrete-time frequency envelope function. The discrete-time HAS model which linearly combines N_h harmonics to synthesise the sequence $y(n)$ is therefore given by:

$$y(n) = \sum_{k=1}^{N_h} A_k(n)\cos(\phi_k(n))$$

$$\phi_k(n) = 2\pi T k \sum_{m=1}^{n} F_0(m) + \Phi_k(n) \tag{2.3.6}$$

For constant fundamental frequency denoted by F_0, we have $F_k(n) = kF_0$ and so $\sum_{m=1}^{n} F_k(m) = \sum_{m=1}^{n} kF_0 = nkF_0$. Hence $\phi_k(n) = 2\pi kF_0 nT + \Phi_k(n)$ in Eq. (2.3.6) for this condition. The PAS model expressed in Eqs. (2.3.5) represents a general sinusoidal additive synthesis model with the HAS model given by Eqs. (2.3.6) representing a special case when partial frequencies are constrained to follow an exactly harmonic distribution.

By definition, WLS is concerned with precisely periodic signals and is therefore described by an *harmonic* spectrum model. Hence, wavetable values are pre-computed "off line" by evaluating Eqs. (2.3.6) with *constant* $A_k(n) = A_k$ and $\Phi_k(n) = \Phi_k$ parameters as illustrated in Figure (2.3.2). The sequence time index, n, becomes analogous to the wavetable sample address and is restricted to span the length of the wavetable. Accordingly, each wavetable lookup operation represents the linear combination of N_h harmonics at a particular phase point thereby providing an efficient implementation of the HAS algorithm. Redefining the constant amplitude and phase vectors (A_k and Φ_k) with an additional index that selects a particular wavetable within a set supports the sequential WLS model outlined in section (2.2.2). We then evaluate Eqs. (2.3.6) with the array variables $A_{k,m}$ and $\Phi_{k,m}$ where m denotes the wavetable index variable with $m \in [0, N_w - 1]$ for N_w wavetables in the set. We develop WLS further in Chapter 4.

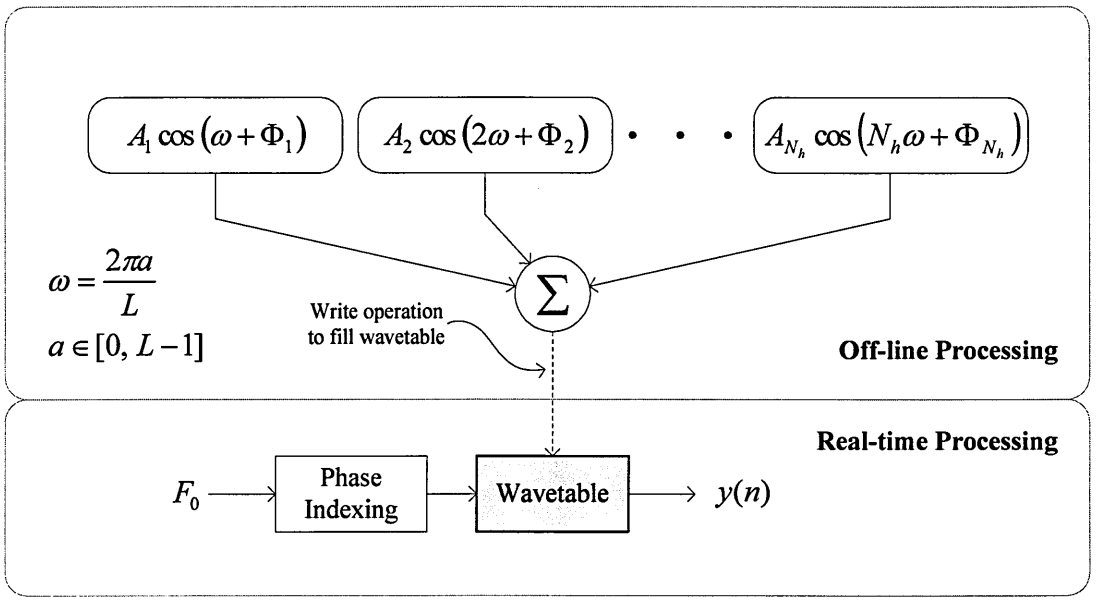


Figure (2.3.2): Simplified harmonic additive synthesis (HAS) processing model using pre-computed wavetable lookup and showing the partition between real-time and "off-line" processing. (Wavetable length and sample address are denoted by L and a, respectively.)

Conversely, the PAS processing model according to Eqs. (2.3.5) and illustrated in Figure (2.3.3) is computationally expensive since each partial must be synthesised and processed individually. The computational advantage afforded by the use of lookup tables implicit in Eqs. (2.3.6) cannot be exploited to simplify the manifold oscillator bank implicit in Eqs. (2.3.5). (We see that PAS as depicted in Figure (2.3.3) can be considered as the limiting case of the GAS model where each wavetable contains a single sinusoid.)

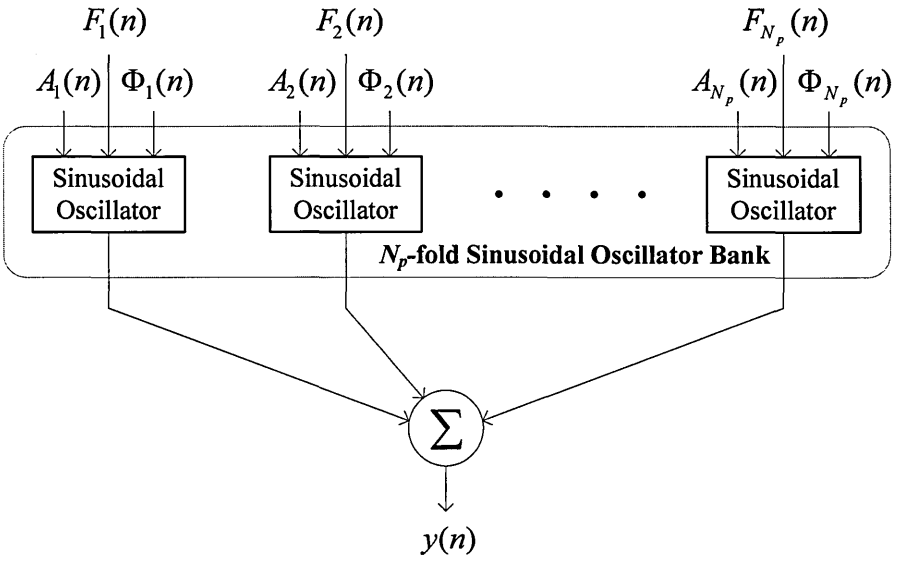


Figure (2.3.3): Partial additive synthesis (PAS) of periodic and non-periodic sounds by time-varying linear combination of N_p partials. The DC component is assumed zero and $\Phi_k(n)$ is typically replaced by a fixed phase offset, Φ_k .

A significant problem with PAS stems from the computational bandwidth required to process parametric control data – the so-called "control parameter problem" [Roads, 1996]. The problem can be stated succinctly, thus:

"*how do we generate and process the enormous amount of data needed to accurately represent a large number of partial envelopes?*"

A prevalent solution reported widely in the literature uses the concept of *source coding*, which represents partial envelope trajectories obtained from the spectrum analysis of natural instruments by approximations which *subjectively* satisfy the so-called identity[11] property upon resynthesis [Grey, 1975; Moore, 1990]. The approximated envelopes provide a baseline spectrum for the resynthesis process. Accurate representation of transient spectral features in the attack region of a sound and prevention of objectionable modulation sidebands in the synthesised spectrum requires that envelope parameters be updated at the same sample rate as the synthesis process. Each partial requires computation of the corresponding $A_k(n)$ and $F_k(n)$ envelopes, with N_p partials therefore requiring $2N_p$ envelope computations *each* sample period. To illustrate the scale of the problem we assume a hypothetical AS processor computing a 100 voice ensemble with each voice composed of 100 partials. Hence we have $N_p = 10^4$ which requires an envelope computation rate of 960×10^6 samples/sec for a 48 kHz sample rate and represents a significant computational imposition.

The psychoacoustic properties of human auditory perception such as *frequency masking* can be exploited via an auditory model to reduce the total partial count and hence envelope computation cost [Marentakis and Jensen, 2002; Jensen, 1999; Moore, 1990]. The *critical band* is an important concept underpinning auditory perception modelling and corresponds approximately to the width of the region along the basilar membrane that is excited by a single frequency sinusoid. This critical bandwidth is approximately 15% of frequency, except at lower frequencies where it increases [Moore, 1990]. Masking effects arise when two or more tones are heard simultaneously, the effect

[11] Resynthesis using suitable partial envelope approximations that produces a sound *subjectively indistinguishable* from the original analysed sound defines the identity property.

reducing with increasing frequency separation between the tones. In general, lower frequencies tend to mask higher frequencies more effectively than vice versa and two single-frequency tones with overlapping critical bands mask each other quite effectively since they cannot be perceived *individually*. Hence, under suitable conditions a *single* tone may be substituted in place of two (or more) tones with no perceptible change. Considering the perception of a complex tone, it is reported that low frequency partials tend to mask high frequency partials and higher frequency partials which fall within a critical band are not perceived *independently* [Moore, 1990]. Exploiting the perceptual abilities of the listener in the identification of superfluous partials is generally known as *receiver coding* in the literature [Moore, 1990]. This technique selectively "prunes" partials that are masked by more prominent ones within a critical band by using a scheduling algorithm that controls the allocation and de-allocation of partials as the parameter envelope values evolve over time. However, computational savings due to a reduced number of partial oscillators must *exceed* the scheduling computation overhead for this technique to be cost effective.

AS is not restricted to the linear combination of basis sinusoids which reflect the Fourier transform. Time-varying linear combination of multiple complex waveforms is reported extensively in the literature and has been outlined in section (2.2.2) as a WLS embodiment within the processed recording subclass. However, this technique is relevant here since the wavetable contents are computed from a spectrum model (i.e. frequency domain specification).

2.3.3 Baseline Spectrum Representation

Risset [1985] reported the first "spectrum analysis driven" AS of trumpet tones in 1964 using the Music V programming language. This pioneering work appears to have included the first application of a piecewise-linear (PWL) envelope approximation to

compress partial envelope trajectories and thereby reduce the control parameter computational overhead [Smith, 1991]. The phase vocoder originally developed at Bell Laboratories in connection with speech synthesis research has provided analysis support to the HAS model for many years [Moorer, 1978; Smith, 1991]. The PARSHL programme has extended the phase vocoder to support the analysis of non-harmonic and pitch-changing sounds, thereby providing partial frequency envelope data and adding greater realism to the synthesis of natural sound classes [Smith and Serra, 1987]. Grey and Moorer [1977] have established the utility of analysis-driven AS in the creation of natural instrument sounds including oboe and clarinet that are subjectively indistinguishable from the original based on listening tests. In particular, this work reports envelope data compression factors of 50:1 using PWL representation, with a 100:1 reduction reported by Serra and Smith [1990] in similar research.

The generality of PAS stems from the inherent accessibility to the elemental components of a sound's timbral composition (i.e. the partials). The penalty for this generality is the need to define a large number of envelope trajectories which control the evolution of partial amplitudes and frequencies that are individually perceptually weak. Beginning the synthesis process *from scratch* is counterintuitive and time consuming without a "baseline" sound spectrum to build from. The efficacy of AS therefore depends on the availability of baseline spectrum data obtained from a spectrum analysis of natural sounds. This analysis produces a weighted set of basis components appropriate to the synthesis model and represents the starting point for creating new sounds and timbral structures.

Various audio spectrum analysis techniques are reported in the literature, including *pitch-synchronous analysis* [Risset and Mathews, 1969], the *phase vocoder* [Serra, 1997] and *constant-Q analysis* [Roads, 1996]. All are based on Fourier analysis

principles and generate large amounts of spectral envelope data with musical signals. Effective implementation of the analysis-resynthesis model requires that spectral envelope data is compressed into a form that permits intuitive parameter editing prior to resynthesis, yet preserves perceptually salient features. The objective is not to save on storage requirements but to ensure intuitive representation and manipulation of the compressed spectrum envelopes and to reduce computational overhead associated with envelope resynthesis. Figure (2.3.4) illustrates a conceptual analysis-resynthesis spectrum modelling environment which accommodates sinusoidal and non-sinusoidal basis functions where the extraction of non-sinusoidal basis functions is peculiar to the MWS subclass. For the PAS model, basis function extraction is not needed since sinusoidal basis functions are assumed.

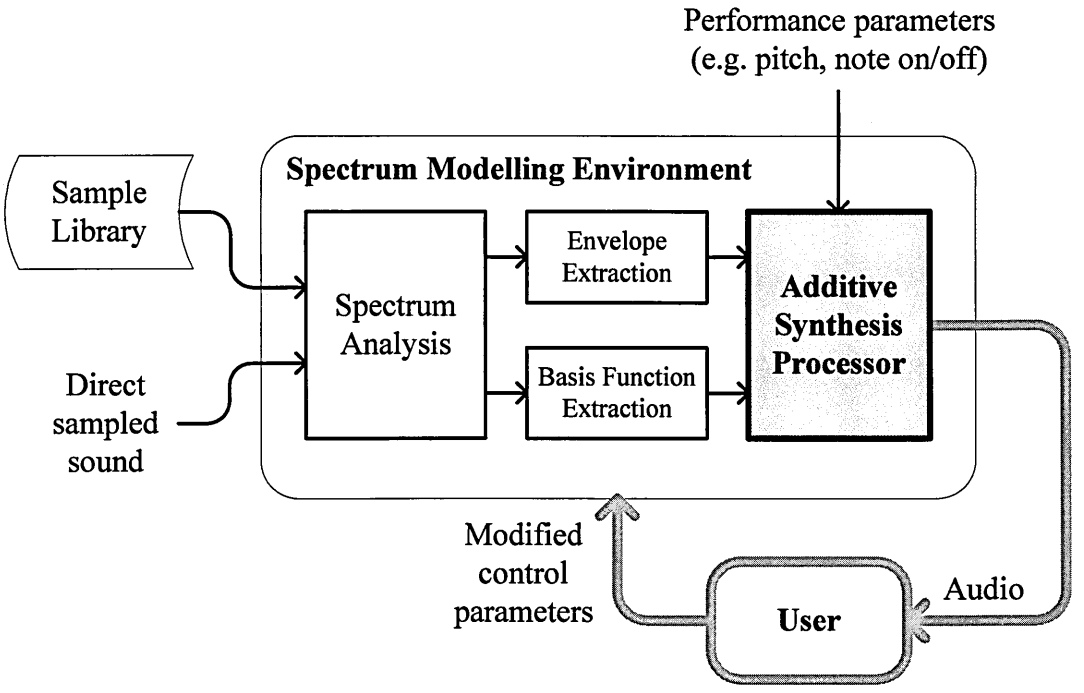


Figure (2.3.4): Top-level information flow in spectrum modelling AS. Analysis of natural sounds generates baseline parameters which are modified to create new sounds using AS.

Spectrum data compression alleviates the control parameter problem and is extensively reported in both the signal processing and computer music literature. The reader is referred to the extensive bibliography reported in Roads [1996]. However, four compression techniques are prevalent: *piecewise-linear* (PWL) *envelope approximation*, *principal components analysis* (PCA), *spectral interpolation synthesis* (SIS) and *spectral modelling synthesis* (SMS). It is evident that each of these techniques is associated with a particular AS model requiring both sinusoidal and non-sinusoidal basis functions.

PWL envelope approximation represents an important spectrum modelling paradigm that simplifies visual representation and reduces envelope resynthesis time [Grey and Moorer, 1977]. Envelopes are represented by PWL approximations of raw analysis data, that is, breakpoints connected by line segments. The technique is not confined to partial or harmonic additive synthesis and finds utility where any complex time-varying parameter requires simplified representation.

Principal components analysis (PCA) reduces a complex waveform into a set of so-called *principal components*, defined as the eigenvectors corresponding to the largest eigenvalues of the covariance matrix [Horner *et al*, 1993]. PCA generates a set of basis waveforms (the principal components) and a corresponding set of weighting coefficients. Linear combination of the basis waveforms with their respective weightings produces a close approximation of the original waveform within some error bound. The principal utility of PCA lies in the generation of MWS basis and weighting data [Horner *et al*, 1993; Sandell and Martens, 1992].

Spectral interpolation synthesis (SIS) is based on time domain linear interpolation between wavetable pairs whose spectra are usually constrained to have corresponding harmonics in phase. Wavetable samples are computed from a HAS model based on the

analysis of natural sounds with phase information discarded [Serra and Smith, 1990]. Frequency domain wavetable specification using Eqs. (2.3.6) is optimal since harmonic phase can be normalised across a wavetable set causing interpolation between corresponding spectra to be free from harmonic amplitude nulls and intuitive in terms of perceived timbral change with interpolation parameter [Smith, 1991; Chamberlin, 1985]. Chamberlin [1985] observes that linear interpolation between two wavetables whose respective harmonics are "phase-normalised" (i.e. corresponding harmonics in each wavetable are in phase) produces a corresponding linear change in harmonic amplitude as the interpolation proceeds.

Spectral modelling synthesis (SMS) decomposes analysed data into *deterministic* and *stochastic* components. The deterministic component is a compressed version of the analysis that preserves the most prominent partials which are then resynthesised using a PAS model. The stochastic component is a noise-like signal and represents the difference between the deterministic component and the original signal computed in the frequency domain [Serra and Smith, 1990]. The synthesised sound, $y(n)$, is represented by the sum of a weighted partial series and a noise signal, $e(n)$, which represents the stochastic component, thus:

$$y(n) = \sum_{k=1}^{N_p} A_k(n) \cos\left[2\pi T \sum_{m=1}^{n} F_k(m) \right] + e(n) \qquad (2.3.7)$$

$A_k(n)$ and $F_k(n)$ are obtained from an analysis and compression process reported in Serra and Smith [1990]. The stochastic component is synthesised by time-varying filtration of a white noise signal using a response which reflects the spectral characteristics of the frequency domain stochastic representation reported in Serra and Smith [1990]. SMS advantages are primarily twofold: data compression associated with the deterministic component envelope representation and synthesis of the stochastic

residual by filtered white noise, thereby saving compared to "brute force" computation using the PAS model. SMS reduces the computational cost of PAS by replacing numerous partials and their parameter envelopes with a *single* filtered noise component.

2.3.4 Subtractive Synthesis

Subtractive synthesis is a spectrum modelling technique based on the application of linear filtering to transform the spectrum of an input signal (or combination of signals) to produce a signal with perceptually desirable timbral properties. Dynamic and parametric timbral articulation are effected by appropriate time-varying parameterisation of the filter frequency response and excitation signal characteristics. Classical analogue synthesis systems are based on the subtractive synthesis model implemented in the analogue domain [Moog, 1965; Chamberlin, 1985].

The subtractive synthesis processing model is illustrated in Figure (2.3.5) and comprises an excitation source feeding a linear filter with both elements having time-varying control parameterisation.

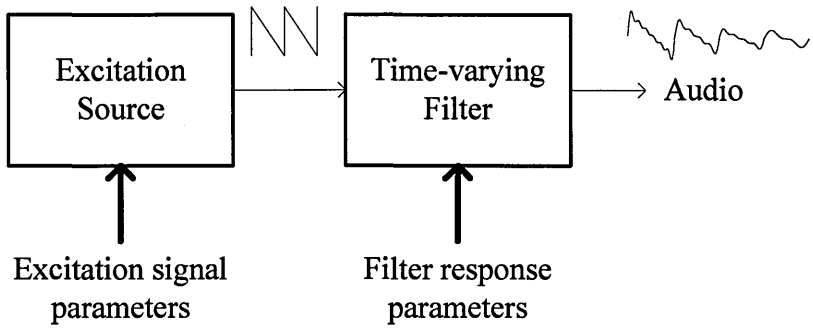


Figure (2.3.5): The subtractive synthesis processing model.

This model aligns well with the generic model of traditional acoustic musical instruments and is closely related to the technique of *physical modelling* discussed later. A violin string coupled through a bridge to a sound box represents a good example [Moore, 1990]. The bowed string generates an excitation signal with time-varying

spectral properties according to the bowing and fingering technique of the player. The bridge couples this sound to a resonant sound box whose frequency response is essentially time-invariant (at least to a first order) but will typically vary among instrument types according to geometry, constituent materials and construction technique. Brass and woodwind instruments (e.g. the trumpet and clarinet) exemplify time-variant acoustic resonators excited by vibrations from the players lips or a reed.

The excitation signal model is typically a harmonic-rich waveform or broadband white noise signal and must contain energy at all frequencies required in the synthesised sound. Subtractive synthesis is unable to *add* energy at a particular frequency [De Poli, 1983]. In some implementations (e.g. sampling synthesis) the excitation signal is provided by a pitch-shifted recorded instrument sound [Roads, 1996].

In general, filter response parameters (e.g. bandwidth) are intuitive and strong. For example, the bandwidth of a low-pass filter exemplifies a strong parameter whose variation controls the perceived "timbral brightness" of the processed signal. The generality of a subtractive synthesis model is constrained by the control flexibility of the digital filter frequency response. We therefore briefly review digital filters and their utility within the subtractive synthesis processing model.

There are broadly two classes of digital filter: *finite impulse response* (FIR) and *infinite impulse response* (IIR) [Orfanidis, 1996]. FIR filters are unconditionally stable and can provide a linear phase response which preserves the time-alignment of all frequency components in the filtered signal. IIR filters are computationally more efficient than FIR filters for a given response characteristic, but do not provide a linear phase response and suffer sensitivity to computation round-off errors inherent with quantised arithmetic leading to instability or limit-cycle behaviour [Rabiner and Gold, 1975]. Effecting a *well behaved* time-varying frequency response is a significant problem in discrete-time

subtractive synthesis. Direct linear interpolation of the filter coefficients produces a corresponding frequency response-time profile which *does not* correspond with that expected [Moore, 1990]. A time-varying FIR filter may be implemented by interpolating the filter response in the *frequency domain* and then transforming to sets of impulse response coefficients using the inverse DFT but it is a computationally expensive technique. Moore [1990] suggests the utility of the two-pole IIR resonant filter which requires only four coefficients and hence four multiplications in the recursive computations. An IIR resonator with normalised peak gain is defined by the transfer function and corresponding difference equation:

$$H(z) = \frac{(1-r)(1-rz^{-2})}{1-2r\cos(\theta)z^{-1}+r^2z^{-2}} \tag{2.3.8}$$

$$y(n) = G\big(x(n) - rx(n-2)\big) + b_1 y(n-1) + b_2 y(n-2)$$

where the centre frequency, f_c, is determined by the pole angle, θ, and the bandwidth, B, by the pole radius, r. Figure (2.3.6) illustrates the corresponding signal-flow architecture.

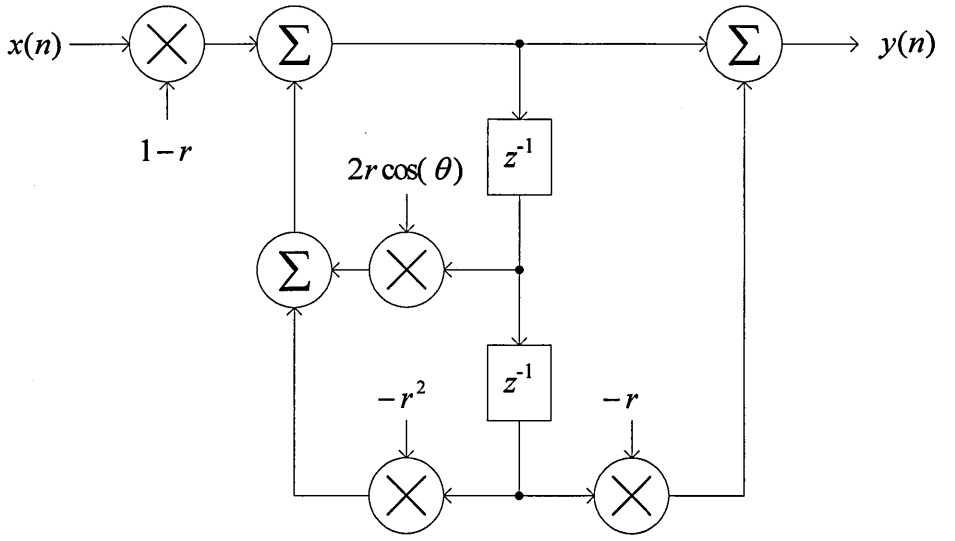


Figure (2.3.6): An IIR resonant filter with normalised peak gain.

The difference equation in Eqs. (2.3.8) requires four coefficients: $r = e^{-\pi BT}$, $G = 1 - r$, $b_1 = 2r\cos(2\pi f_c T)$ and $b_2 = -r^2$, where T is the sample period. In general, we cannot interpolate the coefficients and expect the corresponding frequency response to be well behaved as the interpolation proceeds. Furthermore, pole position must be maintained inside the unit-circle to ensure filter stability. Linear combination of *multiple* two-pole resonators as illustrated in Figure (2.3.7) synthesises a "quasi-arbitrary" filter response according to the filter parameters and their time-varying relative amplitude weightings.

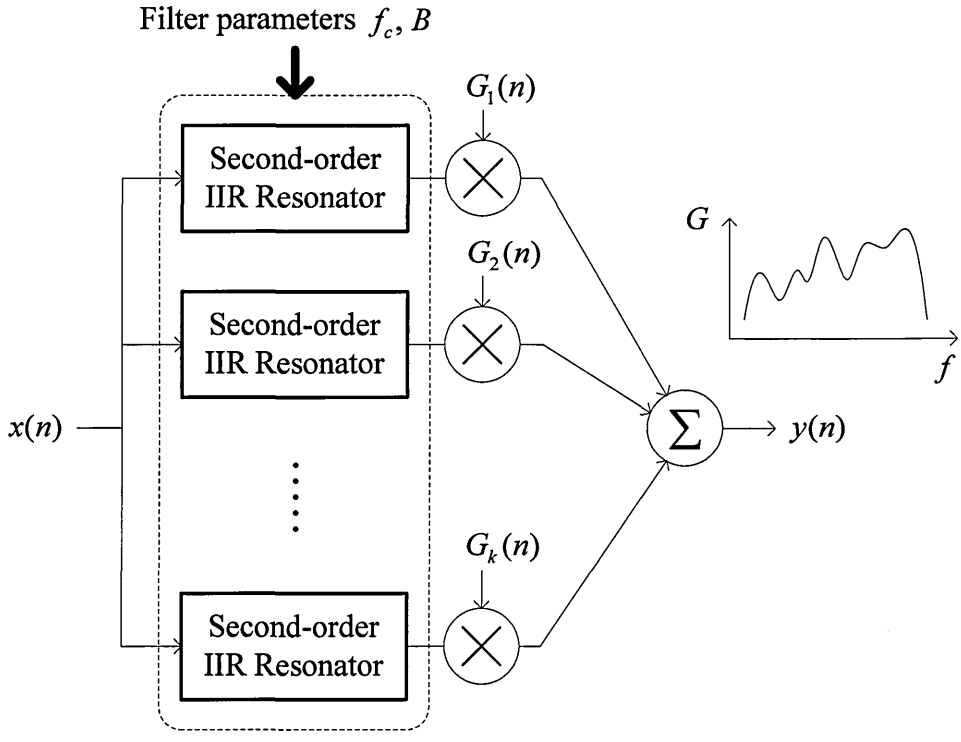


Figure (2.3.7): Weighted linear combination of multiple second-order resonant filter sections. Section weighting is set by the time-varying gain parameter, $G_k(n)$.

Despite the flexibility afforded by this filter architecture within a subtractive synthesis model, it lacks the generality promised by the PAS model. Subtractive synthesis exploits the intuitive correspondence between a frequency domain parameterisation and auditory perception. As a synthesis technique we conclude it is limited compared to the generalisation promised by PAS. However, subtractive synthesis lays the foundations of

physical modelling and thereby precise synthesis of certain instrument classes which conform to the "excited resonator" model.

2.4 Physical Modelling

Physical modelling constructs a mathematical model that describes the salient sound generation mechanism in an instrument which conforms to the "excited resonator" model. The model is typically composed of several distinct blocks which are mutually interactive and in some cases non-linear. Mapping the continuous-time model into the discrete-time domain with quantised values generates the physical model algorithm which is then executed in real time to generate sound samples. The timbral *character* of the simulated instrument is determined entirely by the model structure and not by parametric control. Control parameterisation is closely correlated with the physical parameters of the real instrument and therefore supports intuitive control of the model.

Physical modelling is confined to acoustic instrument structures which are characterised by vibration excitation of a resonant structure (e.g. a guitar string and sound box) [Roads, 1996; Smith, 1992]. The resonant model is typically broken down into several blocks which correspond to physically separate elements of the acoustic instrument. Interconnection between blocks provides access points which serve as points of connection to other parts of the model, excitation inputs and extraction points for the sound samples.

The simplest physical model is the Karplus-Strong plucked string model where the output of a digital delay line, initialised with a pseudo-random noise sequence, $x(-k)\cdots x(-1)$, is fed back to the input via a low-pass filter, $h(n)$, as illustrated in Figure (2.4.1) [Karplus and Strong, 1983].

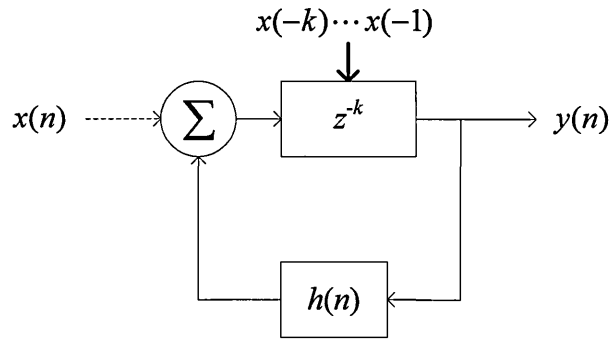


Figure (2.4.1): The Karplus-Strong plucked string model.

Following initialisation, the low-pass filter progressively attenuates high frequency components in the noise sequence which decays to a sinusoid as the recursion proceeds. The steady-state frequency is determined by the length of the delay line. This algorithm is reported as giving excellent simulations of plucked string sounds [Smith, 1987; Smith, 1991] with relatively small computational overhead.

A development of physical modelling known as *waveguide synthesis* is based on the analytical solution of the wave equation that describes the propagation of perturbations in a medium [Smith, 1987]. The *digital waveguide* which underpins this synthesis technique is represented as a pair of delay lines which model the bidirectional propagation of a wave in a lossless medium. Waveguide synthesis uses two-dimensional waveguide meshes connected by lossless *scattering junctions* which model propagation medium stiffness and signal dispersion at a discontinuous junction. Other physical structures are accurately represented by filters as in physical modelling, hence there is a close correspondence between waveguide synthesis parameterisations and our physical perception of acoustic systems. Figure (2.4.2) illustrates a simplified waveguide model of a woodwind instrument [Smith, 2004]. The nonlinear scattering junction simulates the *reed* excitation signal accounting for reed stiffness and embouchure. The *bell* at the end of the clarinet is modelled as a filter. Low frequencies are reflected back into the

bore according to $R(z)$ with high frequencies passed out of the bore according to $H(z)$ providing the output of the model, $y(n)$.

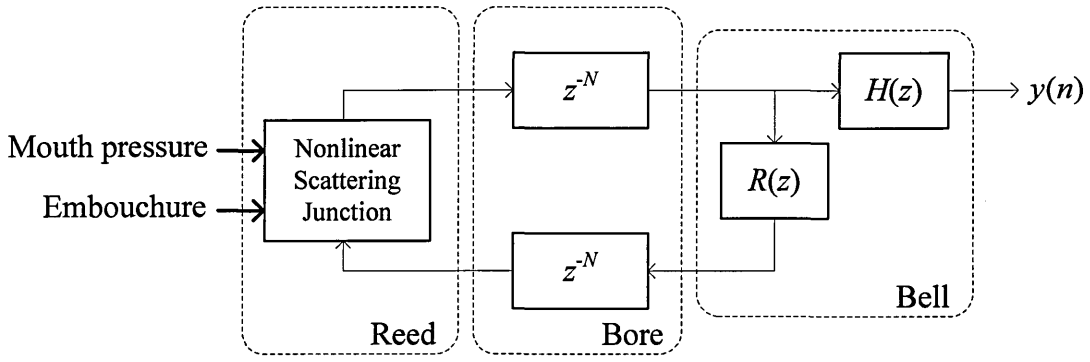


Figure (2.4.2): Simplified waveguide model of a woodwind instrument (e.g. the clarinet).

Waveguide synthesis is reported as successfully modelling complex acoustic systems such as the bore of a clarinet or groups of coupled strings in a guitar, where the guitar bridge represents a resistive coupling mechanism [Borin *et al*, 1997].

The advantages of physical modelling synthesis are the highly physical parameterisations which correspond exactly with those used by the natural instrument (e.g. the bow pressure and bow velocity in a violin physical model [Smith, 1992]) and the robustness of the synthesised sound's identity. For example, Jaffe [1995] observes that the extended Karplus-Strong plucked string algorithm always synthesises plucked string sounds, irrespective of parameter settings in the model [Karplus and Strong, 1983; Jaffe and Smith, 1983]. Physically relevant parameters such as pick position, string flexibility and string thickness can be varied to provide a rich lexicon of sounds which never lose their string-like identity. In contrast, PAS parameters (e.g. partial amplitude) are not directly relevant by themselves when synthesising a plucked string, for example. Physical models with non-linear feedback can exhibit extreme sensitivity to initial condition parameters leading to undesirable chaotic behaviour [Jaffe, 1995].

Physical modelling lacks the generality promised by PAS but provides accurate synthesis and intuitive control of the woodwind and string instrument subclass. The principal disadvantage of physical modelling is that it is fundamentally constrained by the excited resonator model which is appropriate to only a subset of musical instrument classes and corresponding timbres.

2.5 Abstract Algorithm

Abstract algorithm synthesis exploits the properties of certain mathematical functions for synthesising musically useful sounds. *Frequency modulation* (FM) synthesis [Chowning, 1973], Moorer's *discrete summation formulae* [1976] and *waveshaping synthesis* [Risset, 1969] are popular examples of this synthesis technique subclass.

2.5.1 Frequency Modulation (FM) Synthesis

The simplest FM synthesis configuration uses a *carrier* oscillator frequency modulated by a *modulator* oscillator with oscillation frequencies, f_c and f_m, respectively. The modulation depth is controlled by the *modulation index*, $I(n)$, which controls the amplitude of the modulator. The corresponding FM signal is given by:

$$y(n) = A(n)\sin[2\pi f_c nT + I(n)\sin(2\pi f_m nT)] \qquad (2.5.1)$$

where $A(n)$ represents the amplitude envelope. The spectrum of $y(n)$ comprises sidebands surrounding f_c whose amplitudes vary according to k-order Bessel functions of the first kind, $J_k(I(n))$. Eq. (2.5.1) can be expressed in a form which incorporates the Bessel functions directly [De Poli, 1983]:

$$y(n) = A(n)\sum_{k=-\infty}^{\infty} J_k(I(n))\sin[2\pi(f_c \pm kf_m)nT] \qquad (2.5.2)$$

where each k term represents an individual partial. The modulation index controls timbre and for $I(n) = 0$ the spectrum comprises the carrier alone. As $I(n)$ increases, the

spectral envelope describes two peaks symmetrically about f_c which progressively migrate causing partials near f_c to reduce in amplitude and those further away to increase. Therefore, dynamic spectra are realised by varying $I(n)$. The carrier to modulator frequency ratio governs harmonicity, with integer ratios producing harmonic spectra and non-integer ratios producing inharmonic spectra which are useful for synthesising bell-like sounds.

FM synthesis is extendable beyond the two oscillator model [De Poli, 1983]. Six oscillator architectures produce a rich taxonomy of dynamic timbres as exemplified by the Yamaha DX7 synthesiser which has enjoyed huge commercial success [Roads, 1996]. The advantage of FM synthesis lies in the vast range of timbres available with a small set of sinusoidal oscillators, associated modulation arithmetic and their corresponding control parameters [Chowning, 1973]. De Poli [1983] and Roads [1996] provide extensive tutorials on advanced FM synthesis techniques. Disadvantages of FM synthesis lie in non-intuitive control parameterisations (i.e. abstract mathematical variables) that do not correlate well with the audio perception model and a strong "FM timbral identity". However, despite a lack of generality, FM synthesis finds utility due to the timbral range possible for relatively little processing overhead.

2.5.2 Synthesis by Discrete Summation Formulae

Moorer [1976] showed that Eq. (2.5.1) is one instance of a general class of equations called *discrete summation formulae* which provide a computationally efficient method for synthesising band-limited excitation waveforms for the subtractive synthesis model [Moorer, 1976; Moore, 1990]. The expression:

$$y(n) = \sum_{k=1}^{N} \sin k\phi$$

$$= \frac{\sin\left(\frac{N\phi(n)}{2}\right)}{\sin\left(\frac{\phi(n)}{2}\right)} \sin\left[\frac{(N+1)\phi(n)}{2}\right] \tag{2.5.3}$$

describes a DT waveform, $y(n)$, composed of the sum of N equal amplitude harmonics of a frequency, f, where $\phi(n) = 2\pi fnT$. The lower *closed-form* expression requires three multiplications, one division and three table lookup operations as compared to N multiplications, $N-1$ additions and N table lookup operations for the AS form. Moore [1990] observes the singularity arising as $\phi(n) \to m\pi$ for any integer, m, is mitigated by using the identity:

$$\lim_{\phi \to m\pi} \left[\frac{\sin\left(\frac{N\phi(n)}{2}\right)}{\sin\left(\frac{\phi(n)}{2}\right)} \right] = \begin{cases} +N & N \text{ odd} \\ +N & N \text{ even and } m \text{ even} \\ -N & N \text{ even and } m \text{ odd} \end{cases} \tag{2.5.4}$$

However, tabulating the quotient term in Eq. (2.5.3) for a given N in a lookup table indexed by $\phi(n)$ obviates the division operation. Many closed-form summation formulae are given in the literature and some have been applied in generating brass-like tones [Risset and Mathews, 1969]. However, generating excitation waveforms within the subtractive synthesis model is the principal application of this synthesis technique.

2.5.3 Waveshaping Synthesis

Waveshaping synthesis exploits the mathematical concept of *function composition* (i.e. "function of a function"). A function, $f(x)$, is *composable* with another function, $g(y)$, when one function can be used as the argument for the other. Nonlinear waveshaping is concerned with the identification of composing functions that accept

waveform functions as arguments and produce musically useful results. A particular class of composing functions reported in the literature are order-k *Chebyshev polynomials* of the first kind, denoted T_k, that find utility due to their harmonic synthesis property [Moore, 1990]:

$$T_k[\cos(\phi)] = \cos(k\phi) \qquad (2.5.5)$$

A composing function defined as a weighted sum of Chebyshev polynomials maps a cosine wave to a waveform comprising a harmonic series with exactly the same weights as the Chebyshev polynomials. Arfib [1979] showed that a variation in input (argument) frequency yielded *inharmonic* partials in the corresponding output spectrum. However, this technique is necessarily sensitive to the amplitude of the input function. As the input amplitude varies from 0 to 1, the corresponding output spectrum moves from a pure sinusoid (with amplitude that approaches zero) to a spectrum defined by the polynomial weights [Moore, 1990]. Waveshaping synthesis is therefore limited by the strong dependency between output spectrum and the input argument amplitude, although some researchers report techniques for amplitude normalisation [Roads, 1996].

2.6 Generalised Additive Synthesis

The time-varying linear combination of multiple basis components to construct sounds with temporal evolution and parametric articulation of timbre defines the generalised AS paradigm. Basis components are usually sinusoidal although complex waveforms are reported. Harmonic AS is an exact embodiment of the inverse DFT and is therefore constrained to synthesise sounds with sinusoidal basis components whose frequencies are time-invariant and follow an exact harmonic distribution. To emulate the behaviour of natural sounds whose partial frequencies are slowly time-varying, we define the PAS subclass based on linear combination of basis sinusoids with time-varying frequencies,

which may *loosely* follow an harmonic series. We observe that PAS includes the HAS subclass. Figure (2.6.1) illustrates a taxonomy of reported AS forms where two broad AS classes are apparent – *wavetable lookup synthesis* (WLS), as introduced in section (2.2.2) and *direct computation synthesis* (DCS) which describes algorithm execution by brute force computation (e.g. a truncated Taylor's series or the inverse FFT). Both classifications contain the PAS and HAS subclasses, with MWS, GAS and SIS peculiar to WLS. However, implementation of PAS using WLS requires an individual wavetable oscillator for *each* partial.

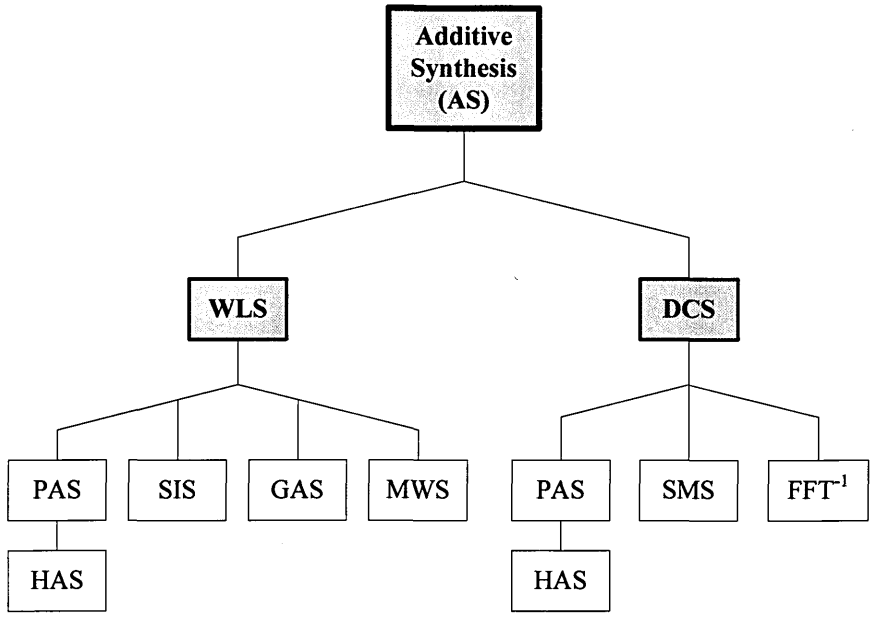


Figure (2.6.1): A taxonomy of additive synthesis subclasses.

The focus of this thesis is AS within the WLS subclass, briefly reviewing pertinent DCS techniques in section 2.6.6 and Chapter 3. We now discuss the PAS paradigm which represents the AS subclass which promises the most generality and flexibility.

2.6.1 Partial Additive Synthesis – Advantages and Disadvantages

The PAS algorithm and methods for its effective implementation are prime motivators for the research reported in this thesis. PAS provides accessibility to the elemental

components of timbral composition and expression through independent control parameters which are consistent with the human auditory perception model. In essence, PAS constructs *directly* the spectrum received along the basilar membrane of the ear [Smith, 1991]. Independent, time-varying control of the frequency, amplitude and phase of each partial is inherent, subject to limitations imposed by computation peculiarities of the implementation method. Furthermore, the number of partials used in the synthesis is bound principally by hardware processing speed.

The manipulation of partial envelope trajectories to construct new sounds is referenced to a known point in "timbre space" by editing a partial envelope set obtained from the analysis of a natural instrument sound. This process is perceptibly intuitive since human auditory perception favours a frequency domain transformation. The weak association between individual partial parameter changes and the corresponding timbral perception remains a problem, however. Baseline envelopes are obtained from a spectrum analysis of natural sounds using the *short-time Fourier transform* (STFT) which provides localisation of frequency and time information. Partial magnitude and phase envelopes emerging from the STFT analysis are converted into corresponding piecewise-linear (PWL) approximated amplitude and frequency envelopes which constitute the baseline parameters.

The principal advantages of PAS include the provision of:

- generality and accessibility of control parameters (i.e. access to the lowest levels of a sound's composition);
- intuitive correspondence with the human auditory perception model;
- temporal evolution of timbre;
- parametric articulation of timbre;

- synthesis baselined against natural sounds via established spectrum analysis tools;
- sequential algorithm architecture conducive to hardware pipelining and therefore "VLSI friendly".

Conversely, the principal disadvantages of PAS include:

- high computation cost associated with synthesising numerous partials;
- high control parameter bandwidth and implicit computation cost;
- weak control parameters;
- dependency on "baseline parameter sets" to initiate the synthesis of new sounds.

We conclude that the advantages of PAS outweigh the disadvantages, excepting weak control parameterisation which remains a fundamental characteristic of the technique. The significant computation overhead motivates investigation of throughput-enhancing techniques (e.g. pipelining) which exploit the sequential structure of the PAS process model to mitigate this problem. This approach is encouraged by the reducing cost-performance ratio of relevant VLSI and memory technology.

2.6.2 The PAS Algorithm – Decomposition and Assessment

Eq. (2.3.5) reveals the fundamentally sequential structure of the PAS algorithm involving two distinct operations: computation of partial phase, $\phi_k(n)$, and the linear combination of N_p partials according to $y(n) = \sum_{k=1}^{N_p} A_k(n)\cos(\phi_k(n))$. The partial phase is defined by:

$$\phi_k(n) = 2\pi T \sum_{m=1}^{n} F_k(m) + \Phi_k(n) \qquad (2.6.1)$$

and may be expressed in an equivalent, but algorithmically more amenable *difference equation* form, thus:

$$\phi_k(n) = \phi_k(n-1) + 2\pi T F_k(n) + \Phi_k(n) - \Phi_k(n-1) \qquad (2.6.2)$$

It is evident from Eqs. (2.6.1) and (2.6.2) that computation of partial phase is a fundamental and irreducible PAS operation.

If we assume the envelope terms are already available, computation of $y(n)$ can be partitioned into five distinct operations:

1. Read the k^{th} partial envelope terms, $A_k(n)$, $F_k(n)$ and $\Phi_k(n)$.
2. Compute the k^{th} partial phase term, $\phi_k(n)$.
3. Compute the k^{th} partial amplitude term, $\cos(\phi_k(n))$.
4. Multiply the k^{th} partial amplitude term by $A_k(n)$.
5. Accumulate the result for $k \in [1, N_p]$.

We ignore for our present discussion that some of these operations can be further decomposed into elemental operations (e.g. computation of $\cos(\phi_k(n))$).

Real-time execution of this algorithm requires a single processor which can compute N_p partials per sample period (i.e. processing *throughput*) with an acceptable *latency* defined as the time between a parametric change (e.g. a change in $A_k(n)$) and the corresponding change in $y(n)$. We postulate that the computational cost of implementing Eqs. (2.6.1) and (2.6.2) is reducible by exploiting their underlying sequential structure within a *pipelined* processor architecture which spreads the computational burden across a sequential processor manifold at the expense of increased latency.

We define *computational cost* in this context as the total number of arithmetic operations required to compute N_p partials per sample period, *per processor*. The "brute force" computational cost, C, (i.e. using a non-pipelined *single* processor architecture) is given by $C = N_p c_p$ operations per sample period, where each partial requires execution of c_p elemental operations. Distributing this algorithm across a p-stage pipeline enables elemental operations to be executed across *consecutive* samples, exchanging throughput for latency. Our rudimentary pipeline comprises p distinct processors each optimised to a specific task, where for the i^{th} stage we execute $C_i = \sum_{i=1}^{p} N_p c_i$ operations per sample period, with c_i denoting the number of elemental operations required to compute one partial. (This representation implies a *non-homogeneous* pipeline where each stage executes a distinct algorithm sub-process with a corresponding variation in the number of elemental operations per pipeline stage.)

The *average* number of operations *per pipeline stage* is given by:

$$C_{ave} = \frac{1}{p} \sum_{i=1}^{p} N_p c_i \qquad (2.6.3)$$

It is clear that as C_{ave} reduces, we require fewer operations to execute per pipeline stage, hence lower cost processing can be utilised for a given N_p, or conversely, a larger N_p can be realised for a given processing speed. C_{ave} reaches a minimum when $c_i = 1$ for all $i \in [1, p]$, whereupon $C_{ave} = N_p$, $C = pN_p$ and $p = c_p$ indicating that *all* elemental operations are pipelined and the pipeline executes one elemental operation per partial per stage with a latency of p sample periods assuming the pipeline is clocked every sample period. Hence we have p pipeline stages each executing N_p elemental operations per sample period (i.e. clocked at $N_p f_s$) in contrast to $N_p c_p$

elemental operations per sample period for the single processor (brute force) implementation.

The pipelined architecture is ultimately constrained by the *slowest* elemental computation time which we denote by $t_m = \max\{t_{op}\}$, where $\{t_{op}\}$ denotes the whole set of elemental operation execution times, hence:

$$N_p c_i' t_m \leq T \tag{2.6.4}$$

where c_i' denotes the number of operations associated with the slowest process (t_m) and is unity in an optimal pipeline, whereupon $N_p t_m \leq T$ or $N_p \leq \frac{T}{t_m}$.

The AS algorithm is inherently a feedforward process (i.e. no global feedback terms) and so we envisage a long computational pipeline, whose latency time, t_l, and hence length is bound only by user perceptual constraints, according to:

$$t_l = pT \leq t_{\max} \tag{2.6.5}$$

where $t_{\max}$ denotes an upper bound imposed by the maximum time that can be tolerated between a control parameter change (e.g. a key depression) and the corresponding auditory perception, typically on the order of 1 ms [Roads, 1996; Alles, 1980]. Since $T \approx 20$ μs, it is evident that $p \leq 50$ and places an upper bound on the pipeline length.

The objective of an AS processor design is therefore to maximise N_p and minimise C_{ave} consistent with the latency time upper bound given by Eq. (2.6.5). We consider AS processing architectures in Chapter 6.

2.6.3 The Significance of Partial Phase

Jensen [1999] reports that early research into the effects of relative partial phase on the human auditory system took two opposing views depending on the auditory model used – the *frequency domain model*, which asserts that phase differences are imperceptible

and the *temporal model*, which asserts that relative phase is perceptible. Perception experiments based on listening tests and summarised in Jensen [1999] conclude that the timbre of musical tones below middle C and the quality of the synthesised human voice depends on partial phase relationships. These conclusions are supported by Quatieri and McAuley [1998] in their work on analysis and synthesis using sinusoidal basis functions.

Risset *et al* [1982] observe that *initial* partial phase is a "perceptually significant" parameter although its effect is weak in a reverberant environment where relative phase relationships become "smeared". Roads [1996] reports that the initial relative phase is particularly important in the perception of attacks and transients, helping to synthesise short-lived components in their correct order. Anderson and Jensen [2001] report psycho-acoustic experiments which indicate the importance of phase information in sound localisation. Results show that phase information is critical to the perception of spatial qualities in the synthesis of binaural sounds.

We conclude that for the synthesis of non-binaural sounds, the *dynamic* phase parameter, $\Phi_k(n)$, in Eqs. (2.3.5) and (2.6.1) can be replaced by a fixed (time-invariant) phase offset, Φ_k, to specify the *initial* phase of each partial. Dynamic phase control via $\Phi_k(n)$ produces a frequency-shifted partial according to the time rate of change of $\Phi_k(n)$. With a static phase offset, dynamic partial frequency envelopes are effected through the $F_k(n)$ parameter.

2.6.4 Piecewise-Linear Envelope Representation

Partial and harmonic additive synthesis requires envelope data obtained from the analysis of natural sounds to baseline the synthesis of new sounds. Partial envelopes obtained directly from the spectrum analysis of natural sounds are generally

characterised by a "noise-like" amplitude variation about a well-behaved, underlying contour. Grey [1975] reasons that the fine detail of partial amplitude and frequency envelope variation is of less subjective importance than the *average* behaviour over the duration of the sound – hence the noise-like variation is perceptually redundant. This hypothesis suggests the utility of an approximate envelope trajectory representation where the superfluous variations are absent leaving only the underlying trend. Piecewise-linear envelope representation uses a line-segment approximation of the underlying trend as exemplified in Figure (2.6.2) for a typical partial amplitude envelope. PWL representation simplifies envelope manipulation and reduces the data bandwidth required to resynthesise envelope trajectories in PWL form.

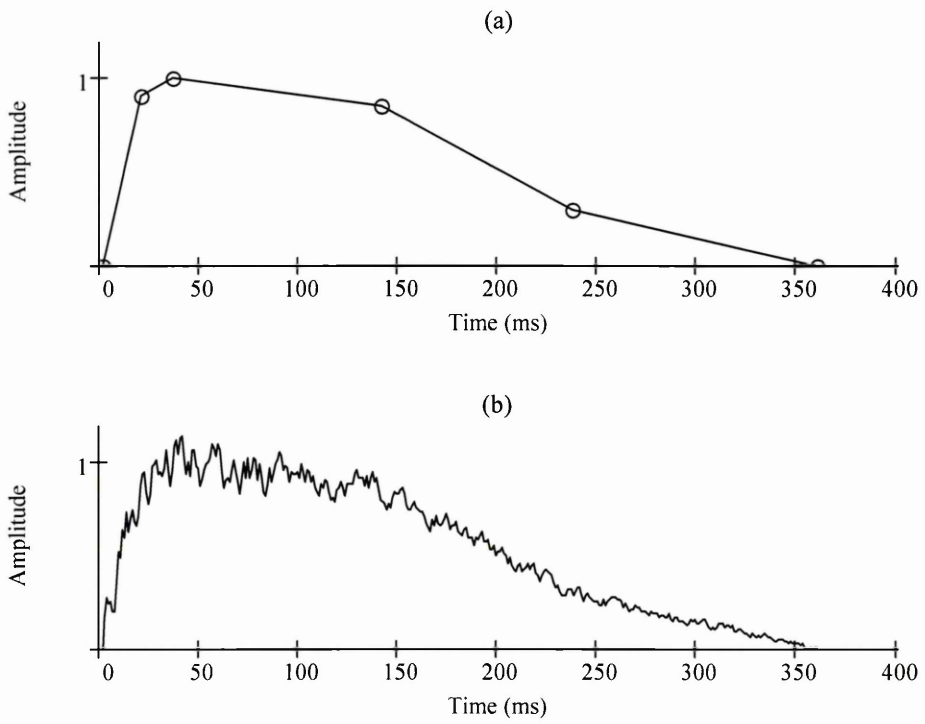


Figure (2.6.2): (a) – Hypothetical PWL approximation of a partial amplitude envelope. (b) – Original envelope exhibiting noise-like variation about an underlying contour.

Grey and Moorer [1977] showed by the use of listening tests that resynthesis using PWL envelope approximation is musically indistinguishable from the original tone and

confirms earlier hypotheses that the noise-like fine detail is largely redundant. The data compression utility of PWL representation is demonstrated when we consider envelopes corresponding to a real musical tone, where only the segment slope and breakpoint threshold information need be stored. Figures (2.6.3) and (2.6.4) illustrate PWL approximations of partial amplitude and frequency envelopes for a trumpet tone [Grey, 1975].

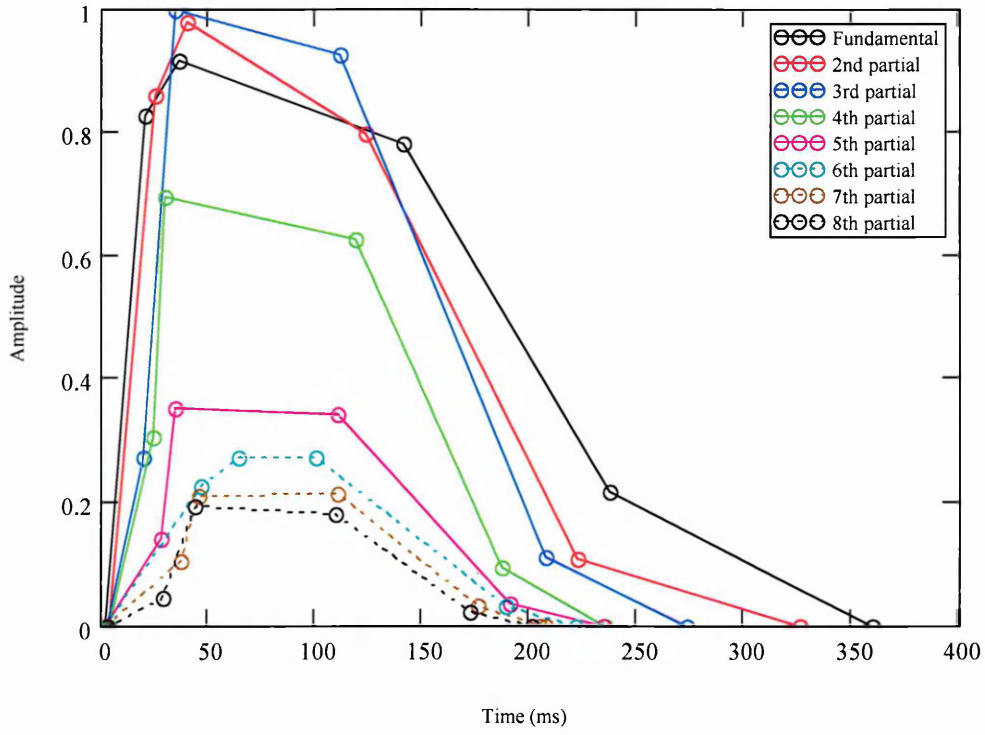


Figure (2.6.3): PWL amplitude envelope approximation of the first 8 partials of a trumpet tone [Grey, 1975].

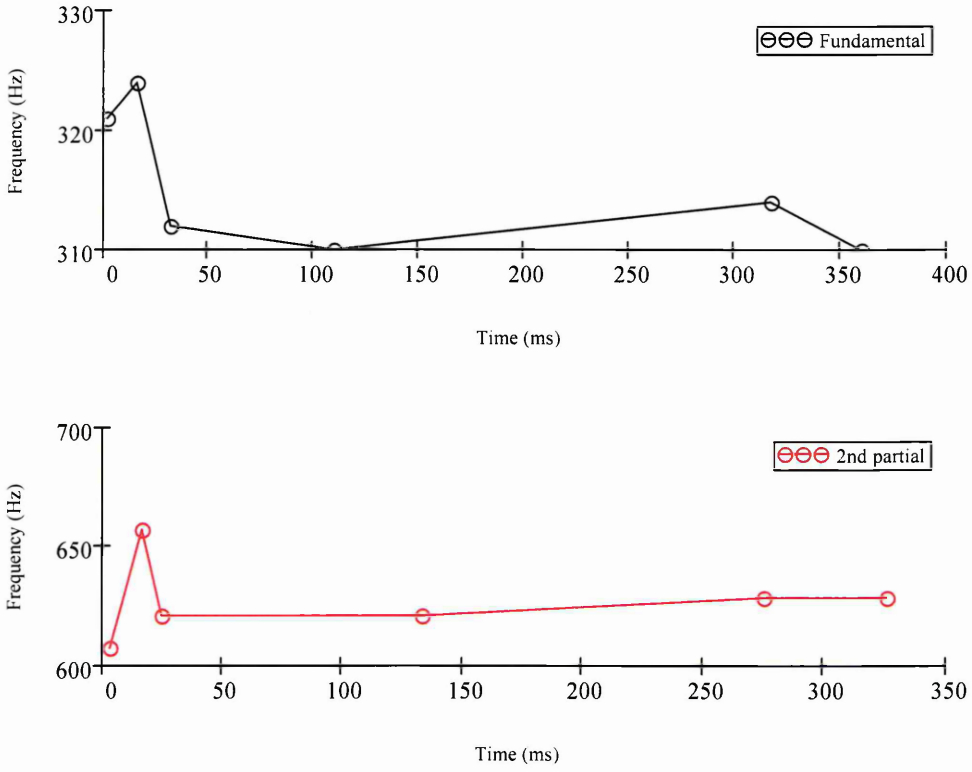


Figure (2.6.4): PWL approximations of the fundamental and 2nd partial frequency envelopes of a trumpet tone [Grey, 1975].

PWL envelopes are stored as a list of breakpoint values corresponding to points of maximum inflection (i.e. stationary points) in the underlying contour. Listed values typically represent *envelope segment slope* and *envelope breakpoint threshold.* Piecewise integration of the listed data effects envelope resynthesis, typically as an integral part of the PAS computation [Snell, 1977].

2.6.5 Metaparameters – Context and the PAS Processing Model

We define a *metaparameter* as a single parameter that modifies a group of partial parameters according to a predefined mapping function [Jaffe, 1995]. Transformation typically involves linear scaling of amplitude and frequency envelopes or a combination of both. Metaparameters map between numerous, individually weak PAS parameters

and a small set of *strong* parameters with the objective of providing intuitive and expressive articulation of timbre.

If we assume that the partial phase envelope is replaced by a fixed constant, Φ_k, we modify Eqs. (2.3.5) to include amplitude and frequency metaparameter scaling terms, $a_k(n)$ and $b_k(n)$, thus:

$$y(n) = \sum_{k=0}^{N_p} a_k(n) A_k(n) \cos\left(2\pi T \sum_{m=1}^{n} [b_k(m) F_k(m)] + \Phi_k \right) \qquad (2.6.6)$$

The $a_k(n)$ terms effect "response shaping" of the synthesised spectrum whereas the $b_k(n)$ terms provide a "harmonicity scaling" of the constituent partials. Since auditory perception of musical pitch is related to the ratio of frequencies, we note that multiplicative scaling of the frequency envelope term is appropriate since partial frequency ratios relative to the fundamental are preserved.

A filter with parametrically-varying frequency response is effected by defining $a_k(n)$ as a function of $F_k(n)$ and additional parameters (or *metaparameters*) which determine the response shape. For example, we define a PWL low-pass response (adapted from Jaffe [1995]) which may be likened to a frequency domain filter specification, thus:

$$a_k(n) = \begin{cases} F_k^{-r_1}(n) & F_k(n) < f_b \\ f_b^{\,r_2 - r_1} F_k^{-r_2}(n) & F_k(n) \geq f_b \end{cases} \qquad (2.6.7)$$

where f_b denotes the breakpoint frequency and $r_1, r_2 \in [0, 1]$ determine the response slope before and after f_b, respectively. The variables r_1, r_2 and f_b represent metaparameters which control frequency response *shape* as exemplified in Figure (2.6.5) and affords a simple "timbral brightness" control.

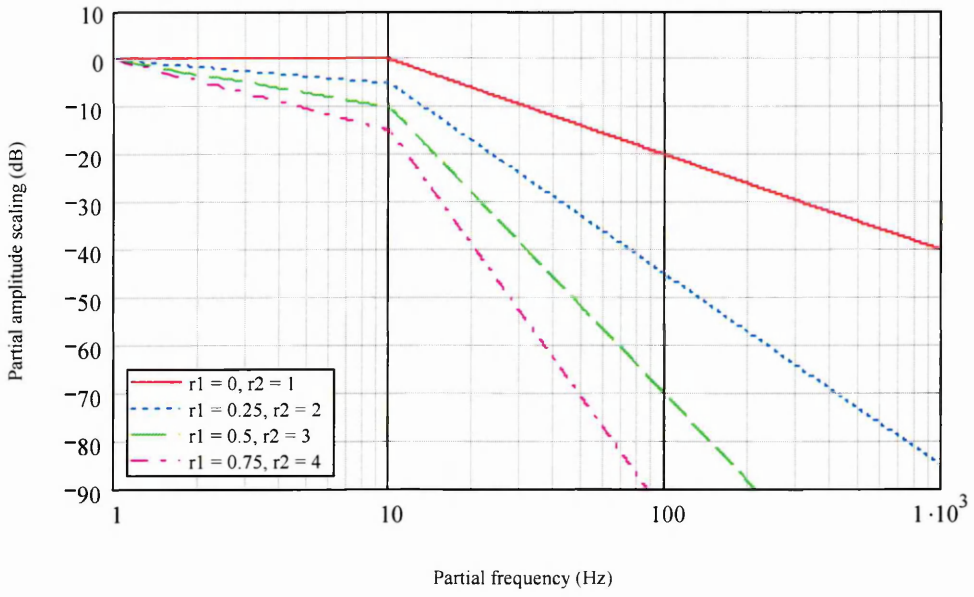


Figure (2.6.5): PWL partial amplitude response for various r_1 and r_2 values. (Response is normalised to a fundamental frequency of 1 Hz with f_b set at 10 Hz.)

Figure (2.6.6) illustrates the PAS processing model incorporating the $a_k(n)$ and $b_k(n)$ metaparameter scaling terms. For N_p partials, this model requires $2N_p$ envelope generators, N_p sinusoidal oscillators, $4N_p$ multiplications, N_p additions and $2N_p$ metaparameter mapping operations *each sample period.*

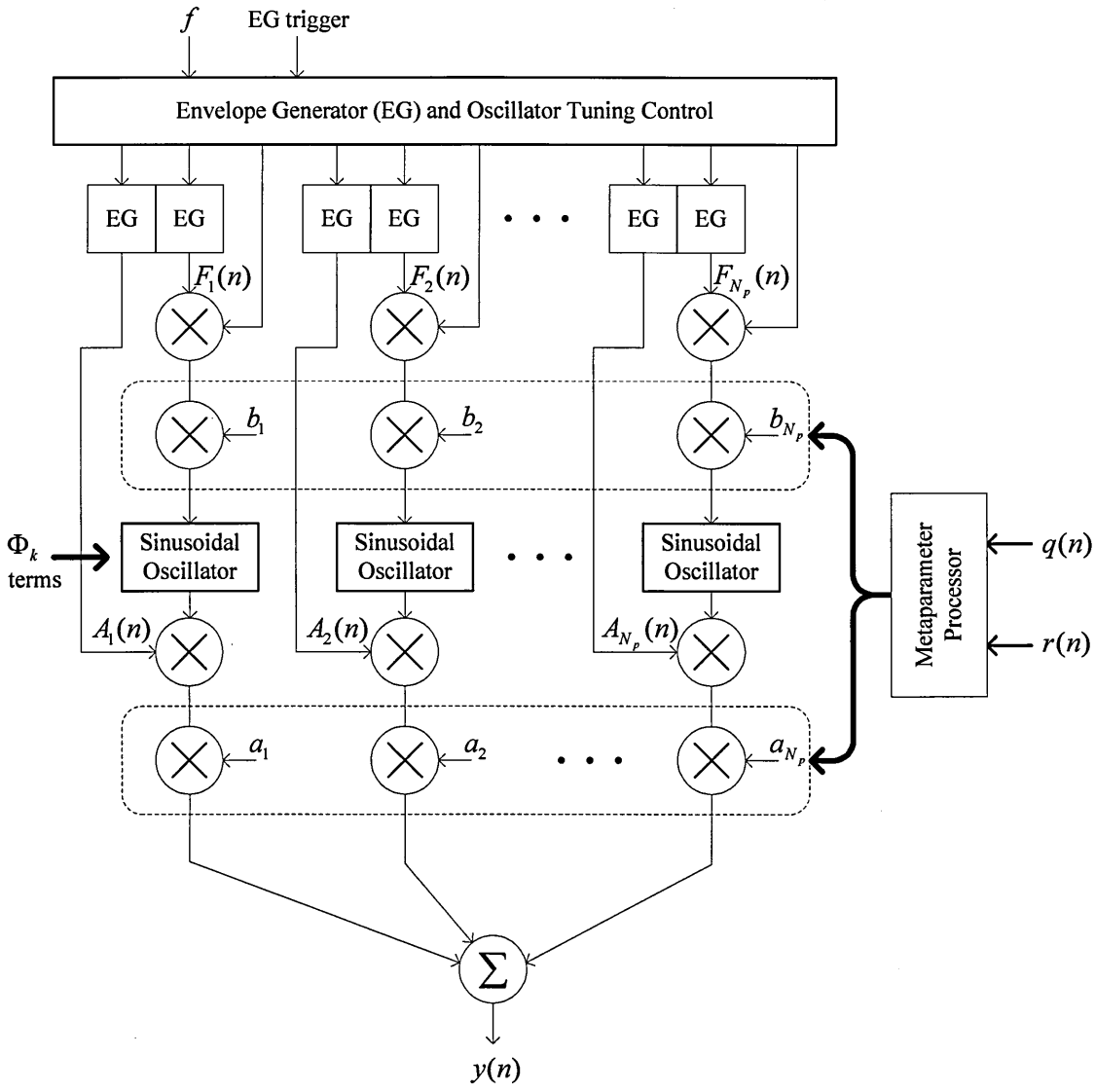


Figure (2.6.6): The partial additive synthesis (PAS) processing model incorporating metaparameterisation of partial amplitude and frequency. The $q(n)$ and $r(n)$ terms denote arbitrary, time-varying metaparameters and the EG blocks represent PWL envelope generators.

2.6.6 Additive Synthesis using the Inverse FFT

The inverse fast Fourier transform (IFFT) is a computationally efficient algorithm for computing the inverse discrete Fourier transform. The IFFT transforms a complex discrete-frequency vector, $X(\omega)$, comprising N locations (or *bins*) into a corresponding discrete-time signal vector, $x(n)$, of N samples. The IFFT is a *block processing* algorithm with each block or *frame* containing N samples which are transformed en bloc. Real-time synthesis requires continuous application of the IFFT to consecutive frames containing complex frequency data obtained from the short-time Fourier transform (STFT) of the desired sound, with N chosen to give acceptable time resolution [Chamberlin, 1985; Roads, 1996]. Parametric control is only permissible at the frame processing rate, $\frac{2f_s}{N}$, where parameter step changes between frames cause undesirable noise in the synthesised signal which can be mitigated by interpolation of amplitude and frequency parameters between frames [Chamberlin, 1985].

The IFFT synthesises the first $\frac{N}{2}$ harmonics of the frame processing frequency. Arbitrary partial frequencies are synthesised by rounding the required frequency to the *nearest* bin frequency and adding a phase offset to the complex frequency term to approximate the *residual* frequency. However, this technique introduces phase discontinuities at the frame boundaries with non-overlapping frames. *Overlap-add* synthesis with raised cosine windows reduces the magnitude of these discontinuities and interpolates the partial amplitude as $A_k(n)$ evolves among frames. However, this technique requires twice the number of IFFT computations and produces objectionable amplitude modulation due to smearing of the phase discontinuities across consecutive frames [Chamberlin, 1985].

Rodet and Depalle [1992] present the "FFT^{-1}" algorithm which mitigates AM effects associated with phase discontinuity smearing, although implementation of partial frequency envelopes is computationally complex. One solution proposed by Goodwin and Rodet [1994] employs frequency "chirps" within frames combined with an overlap-add "splicing" algorithm. Maintaining phase continuity between frames requires computation of a quadratic polynomial to compensate for the parabolic phase contour of the chirp. Goodwin and Kogon [1995] propose further refinements which reduce significant inter-frame splicing errors when the frequency increment is not constant between frames. The computational complexity of the FFT^{-1} algorithm requires carefully coded software implementation to ensure data and instruction fetches are confined to cache memory to maximise execution speed. Freed *et al* [1993] report the synthesis of approximately 320 partials at a sample rate of 44.1 kHz using an optimised C code FFT^{-1} algorithm running on a MIPS R4000 workstation. Similarly, a hypothetical VLSI implementation of a linearly combined sinusoidal oscillator bank using a 50 MHz clock with 4 cycles per oscillator yields approximately 290 partials. In contrast, a software driven DSP running at 50 MHz with 20 clock cycles per oscillator sample yields approximately 56 partials. Finally, Hodes and Freed [1999] report 608 partials synthesised using a direct-form recursive oscillator algorithm executing on the SPERT vector coprocessor [Asanovic *et al*, 1995].

The FFT^{-1} algorithm requires a STFT pre-processing operation which transforms the partial envelopes into a *short-time spectrum* (STS) for subsequent time domain transformation using an IFFT with overlap-add splicing. The associated processing overhead is *independent* of the number of partials. However, computing the STS incurs a computational overhead proportional to the number of partials and leads to an upper bound on the efficacy of the FFT^{-1} algorithm [Phillips, 1996].

2.7 Conclusions

This review suggests a natural partitioning of the research objective according to the AS subclass. Several distinct AS subclasses have been identified, differentiated primarily by basis component characteristics and linear combination methodology. We therefore partition the research objective into distinct topics which collectively align with the AS paradigm and individually define the focal areas of this thesis. We postulate that table lookup operations are faster than direct, brute force computation of the tabulated data and so we consider the WLS subclass of Figure (2.6.1). Moreover, we hypothesise that intrinsic arithmetic partitioning evident in the linear combination of manifold basis components, as common to all AS classes, motivates the utilisation of a systolic pipelined processing architecture to effect algorithm computation. Each pipeline stage is optimised to execute a particular elemental function exploiting table lookup to replace direct computation.

Chapter 3 Digital Sinusoidal Oscillators

3.1 Overview

This chapter presents an original perspective on the application of discrete-time (DT) sinusoid synthesis algorithms, reported in the literature, to multiple-oscillator additive synthesis. We have seen in section (2.3.2) that PAS requires numerous linearly combined sinusoidal oscillators, with each having *independent* control of amplitude, frequency and phase. Each oscillator must provide a constant amplitude, *phase-continuous* frequency transition at any phase point. We define a phase-continuous frequency transition as one where the underlying phase-time characteristic shows only a change in *slope* at the transition point with no step change in phase. It follows that the corresponding amplitude signal will not contain a step change at the frequency transition point, similar to analogue (continuous-time) voltage controlled oscillator (VCO) behaviour. The step amplitude changes which generally accompany phase-discontinuous transitions are perceived as objectionable 'clicks' in the audio signal. We consider phase continuity further in Chapter 4.

The principal objective of this review is to identify an optimal sinusoidal oscillator algorithm using assessment criteria relevant to computer music additive synthesis. Table (3.1.1) summarises six properties (P1 to P6) against which we compare and assess prototype oscillators in an objective manner. The time-varying amplitude, frequency and phase envelopes of the k^{th} partial we define with the parameters $A_k(n)$, $F_k(n)$ and $\Phi_k(n)$, respectively. There are two complementary classes of digital sinusoidal oscillator algorithm – *recursive* and *phase-accumulating* [Tierney *et al*, 1971]. Recursive oscillators are essentially DT simulations of physical (e.g. mass-spring)

oscillatory systems having a simple harmonic motion with zero damping as their solution.

Property	Description
P1	Arithmetic overhead (e.g. number of multiply and add operations)
P2	Suitability to time-division multiplexing
P3	Amplitude stability and spectral purity over time
P4	Interaction between $F(n)$, $A(n)$ and $\Phi(n)$
P5	$F(n)$ response characteristic and dynamic range
P6	Phase-continuous frequency transition

Table (3.1.1): Six key properties of digital sinusoidal oscillator algorithms requiring consideration for optimal application in partial additive synthesis.

In general, the cost of implementation is bound by the number of multiplication operations required per sample, ranging from two to four with recursive algorithms. Interaction between oscillation frequency, amplitude and phase is undesirable since it increases control complexity. This is of particular concern with the direct-form algorithm where, despite computational simplicity, each frequency transition requires re-initialisation with new initial conditions to maintain amplitude and phase-continuity. This research has produced definitions of initial conditions that provide phase-continuous frequency transition (see section (3.2.6)) and this work has been published [Symons, 2004].

Phase-accumulating oscillators allow *independent* sample rate control of $A_k(n)$, $F_k(n)$ and $\Phi_k(n)$, in line with the classical definition of additive synthesis presented in

section (2.3). These oscillators compute the sinusoid phase explicitly from a sample rate integration of $F_k(n)$ and then map to the amplitude domain using a *phase-mapping* function. This function can be effected with a lookup table [Tierney *et al*, 1971] whose length and word size control mapping accuracy between the phase and amplitude domains. (In subsequent discussion we drop the k subscripts for brevity.)

3.2 Recursive Oscillators

3.2.1 Direct-form

The simplest recursive oscillator is based on the direct-form second-order resonator developed from the z-transform pair [Orfanidis, 1996]:

$$h(n) = r^n \sin(n\theta)u(n) \quad \Leftrightarrow \quad H(z) = \frac{r\sin(\theta)z^{-1}}{1 - 2r\cos(\theta)z^{-1} + r^2 z^{-2}} \qquad (3.2.1)$$

where $u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$. The poles of $H(z)$ comprise the conjugate pair $re^{\pm j\theta}$, where r represents the radial distance of the pole from the origin in the complex z-plane. For $r \in (0,1)$, the pole pair describe an exponentially decaying DT sinusoid, $h(n)$, with frequency controlled by θ and amplitude envelope $r^n = e^{n\ln r}$. Setting $r = 1$ places the poles precisely on the unit circle and produces a sinusoidal impulse response with constant unit amplitude for all n.

Since $H(z) = \frac{Y(z)}{X(z)}$, the right hand side of Eq. (3.2.1) can be written as $(1 - 2r\cos(\theta)z^{-1} + r^2 z^{-2})Y(z) = (r\sin(\theta)z^{-1})X(z)$. Taking the inverse z-transform of this expression yields the second-order difference equation:

$$y(n) = 2r\cos(\theta)y(n-1) - r^2 y(n-2) + r\sin(\theta)x(n-1) \qquad (3.2.2)$$

where $y(n)$ represents the DT oscillator output sequence and $x(n-1)$ is a forcing function which initiates oscillation at $n=1$, with initial conditions (IC) $x(-1)=0$ and $y(-1)=y(-2)=0$. The frequency control parameter, θ, is constrained to $\theta \in (-\pi, \pi)$ and with $r=1$ produces a unit amplitude oscillation after the oscillator is initiated. Applying the forcing function $x(n)=A\delta(n)$, where $\delta(n)=\begin{cases}1, & n=0\\ 0, & n\neq 0\end{cases}$, produces an output sinusoid of amplitude A given by $y(n)=A\sin(n\theta)$ for $x(-1)=0$ and $y(-1)=y(-2)=0$. We observe that the impulse input function only serves to initiate the recursive process; thereafter the oscillation is self-sustaining since the system has no damping as the poles lie exactly on the unit circle ($r=1$) in the complex z-plane. The process may be simplified by using the ICs, $y(-1)$ and $y(-2)$, to provide the initiation stimulus eliminating the input term, $x(n)$. Eq. (3.2.2) now becomes:

$$y(n)=2\cos(\theta)y(n-1)-y(n-2) \qquad (3.2.3)$$

with ICs $y(-1)$ and $y(-2)$ at $n=0$. Physical realisation of Eq. (3.2.3) is illustrated in Figure (3.2.1).

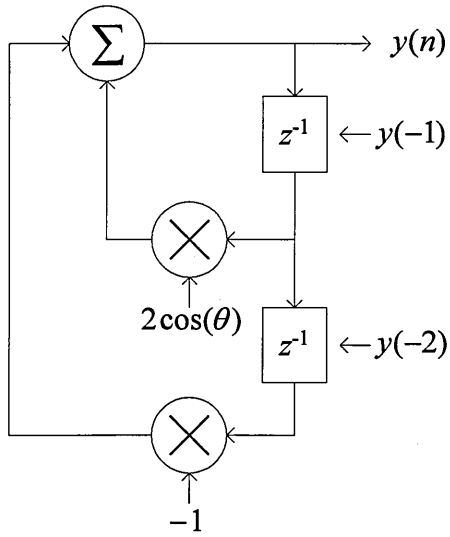


Figure (3.2.1): The direct-form recursive oscillator.

Setting $y(-1)=0$ gives $y(n)=\frac{-y(-2)}{\sin(\theta)}\sin((n+1)\theta)$, which describes a DT sinusoid with amplitude $\frac{y(-2)}{\sin(\theta)}$ and no phase shift term [Abu-El-Haija *et al*, 1986]. With $y(-2)=-A\sin(\theta)$, $y(n)$ describes a sinusoid with amplitude A, and frequency a function of θ. We observe there is no simple definition of $y(-2)$ that provides *independent* control of amplitude and phase for a particular frequency.

We now consider the ICs , $y(-1)$ and $y(-2)$, required to generate the generalised DT sinusoid $y(n)=A\sin(n\theta+\phi)$ for $n\geq 0$. We first consider the z-transform of Eq. (3.2.3) taking account of the initial conditions. We have $Z\{y(n)\}=Y(z)$, $Z\{y(n-1)\}=z^{-1}Y(z)+y(-1)$ and $Z\{y(n-2)\}=z^{-2}Y(z)+z^{-1}y(-1)+y(-2)$, where $Z\{a\}$ denotes the z-transform of a. Thus we obtain:

$$Y(z)=\frac{2y(-1)\cos(\theta)-y(-2)-y(-1)z^{-1}}{1-2\cos(\theta)z^{-1}+z^{-2}} \qquad (3.2.4)$$

The inverse z-transform of equation Eq. (3.2.4) has the general form:

$$\begin{aligned} y(n) &= A\cos(\phi)\sin(n\theta)+A\sin(\phi)\cos(n\theta), \quad n\geq 0 \\ &= A\sin(n\theta+\phi), \quad n\geq 0 \end{aligned} \qquad (3.2.5)$$

and defines a generalised DT sinusoid with amplitude A, frequency a function of θ, and phase ϕ. If we set $\theta=\omega T$, with ω the angular oscillation frequency, we can establish a relationship between $y(-1)$ and $y(-2)$ and the amplitude, frequency and phase parameters of the general DT sinusoid of Eq. (3.2.5). The results follow from comparing Eq. (3.2.4) with the inverse z-transform of Eq. (3.2.5) and equating the coefficients of z^0 and z^1 in the numerators of the two expressions. After some algebraic and trigonometric manipulation we obtain:

$$y(-1) = A\sin(\phi - \theta)$$
$$y(-2) = A\sin(\phi - 2\theta) \tag{3.2.6}$$

If the frequency parameter in Eq. (3.2.3) is changed from θ to θ' (i.e. ω to ω') at some sample index m, the ICs for the 'new' recursion, $y'(-1)$ and $y'(-2)$, will be the last two samples of the recursion with frequency ω, that is $y'(-1) = y(m-1)$ and $y'(-2) = y(m-2)$. The effect, illustrated in Figure (3.2.2), is to produce an *approximately* phase-continuous frequency transition from ω to ω' simultaneous with a step change in amplitude from A to A'. The underlying phase function of $y(n)$ is also shown in Figure (3.2.2) and illustrates the phase-discontinuity at the transition point.

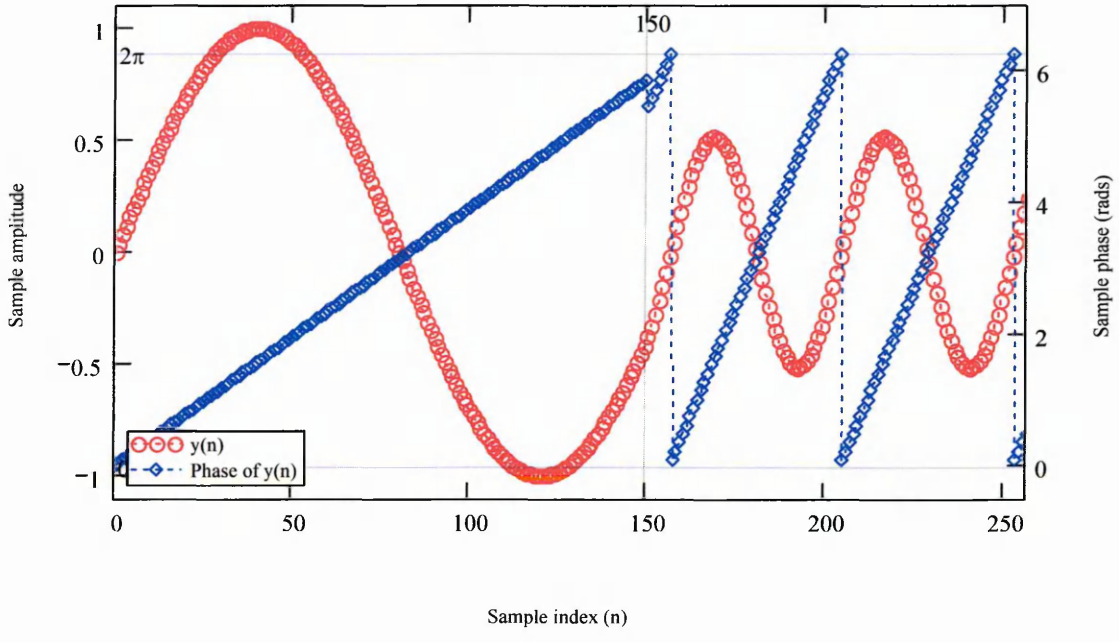


Figure (3.2.2): $y(n)$ for the direct-form oscillator with frequency transition at $n = 150$, showing the normalised phase with phase-discontinuity clearly evident.

The new amplitude is dependent on ω' and the oscillation phase (a function of $y(m-1)$) where the frequency transition occurs. The new amplitude A' is found by eliminating ϕ from Eqs. (3.2.6), yielding:

$$A' = \sqrt{\left(\frac{y(m-1)\cos(\omega' T) - y(m-2)}{\sin(\omega' T)}\right)^2 + (y(m-1))^2} \qquad (3.2.7)$$

We can use $(A')^{-1}$ as determined from Eq. (3.2.7) to normalise $y(n)$ to unit amplitude following a frequency transition. However, this introduces a step amplitude discontinuity into $y(n)$ as shown in Figure (3.2.3) and incurs additional computation of the $(A')^{-1}$ normalising term.

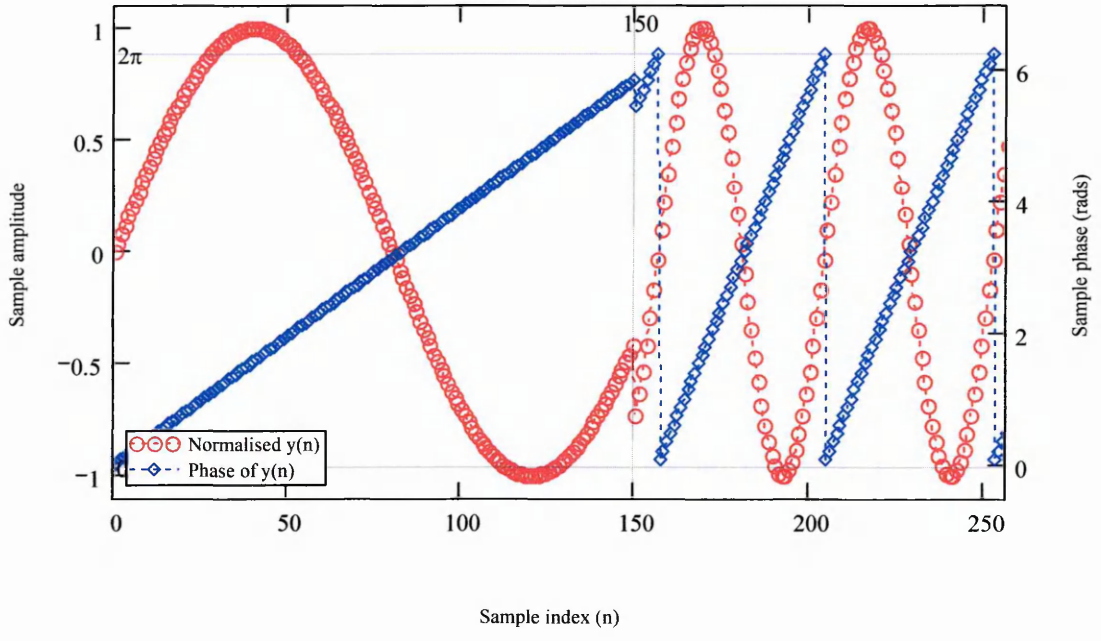


Figure (3.2.3): Normalising $y(n)$ to unit amplitude ($A = A' = 1$) introduces an amplitude-discontinuity at the transition point ($n = 150$).

Inspecting Figures (3.2.2) and (3.2.3) we observe a phase discontinuity at the frequency transition point. The end sample $y(m-1)$, of the recursion with frequency ω clearly has a different phase to the initial sample $y(m)$, of the recursion with frequency ω'.

Generating constant amplitude sinusoids with phase-continuous frequency transitions requires computation of new ICs at *every* frequency transition point. These ICs require knowledge of the oscillation phase just before the transition point which we consider further in section (3.2.6).

We observe that a change in the frequency coefficient produces a corresponding change in oscillation frequency simultaneous with a change in amplitude. It is also evident that the amplitude change is well behaved in an analytic sense since there is no step discontinuity at the transition point. Amplitude normalisation introduces a step amplitude discontinuity at the transition point which is perceived as an audible 'click' and is therefore undesirable. It is not evident that the *phase* discontinuity observed in the underlying phase function of Figure (3.2.2) is perceptible in isolation, but the necessary amplitude normalisation will be. We observe that, in general, larger phase discontinuities produce correspondingly larger step amplitude discontinuities upon normalisation and will therefore be more audible.

3.2.2 Coupled-form

The coupled-form oscillator [Proakis & Manolakis, 1996] is illustrated in Figure (3.2.4) and is characterised by the matrix multiplication of a two dimensional vector described by the matrix difference equation, thus:

$$\begin{bmatrix} y_1(n) \\ y_2(n) \end{bmatrix} = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} y_1(n-1) \\ y_2(n-1) \end{bmatrix} \qquad (3.2.8)$$

where $a = r\cos(\theta)$ and $b = r\sin(\theta)$.

The matrix operates on the vector $[y_1(n-1) \quad y_2(n-1)]^T$ to effect a combined θ rotation and r scaling on each sample event, with ICs $y_1(-1)$ and $y_2(-1)$ obviating the initialisation stimulus $x(n)$.

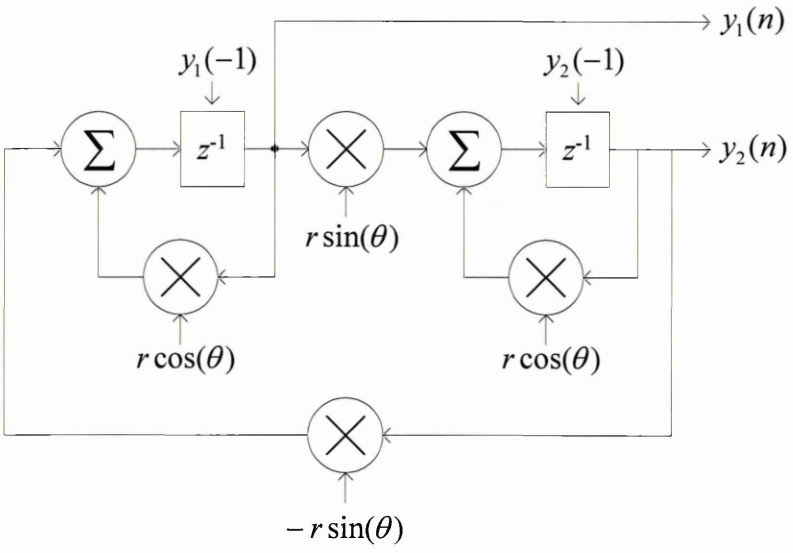


Figure (3.2.4): The coupled-form recursive oscillator.

Setting $r=1$, $y_1(-1)=A\cos(\theta)$ and $y_2(-1)=-A\sin(\theta)$ produces unit amplitude quadrature (complex) sinusoids $y_1(n)=\cos(n\theta)$ and $y_2(n)=\sin(n\theta)$ for $n\geq 0$. The corresponding z-domain transfer functions are given by:

$$H_1(z)=\frac{1-az^{-1}}{1-2az^{-1}+(a^2+b^2)z^{-2}}$$
$$H_2(z)=\frac{bz^{-1}}{1-2az^{-1}+(a^2+b^2)z^{-2}} \qquad (3.2.9)$$

We can factorise the denominators of Eqs. (3.2.9) into $(1-pz^{-1})(1-p^*z^{-1})$, where $p=(a+jb)$ and $p^*=(a-jb)$ represent the conjugate poles of $H(z)$ with $p=re^{+j\theta}$ and $p^*=re^{-j\theta}$. Linearly combining these transfer functions and noting that the numerators combine to give $(1-p^*z^{-1})$ we obtain the pole-zero cancellation [Orfanidis, 1996]:

$$H(z)=H_1(z)+jH_2(z)=\frac{1-p^*z^{-1}}{(1-pz^{-1})(1-p^*z^{-1})}=\frac{1}{1-pz^{-1}} \qquad (3.2.10)$$

The z-plane representation therefore comprises a *single* pole at $re^{j\theta}$, and so the coupled-form oscillator is a first order system. However, maintaining $r = \sin^2(\theta) + \cos^2(\theta) = 1$ for all values of quantised θ is not possible and causes exponential growth or decay of the oscillation sequence when $r \neq 1$. Figures (3.2.5a) and (3.2.5b) illustrate the pole distribution over frequency in the complex z-plane, with $r = 1$ and a coarse quantisation interval chosen to exaggerate distribution effects for clarity. Accordingly, these figures have been obtained by plotting the complex roots of $1 - 2az^{-1} + z^{-2} = 0$ and $1 - 2az^{-1} + (a^2 + b^2)z^{-2} = 0$ for the direct-form and coupled-form oscillators, respectively.

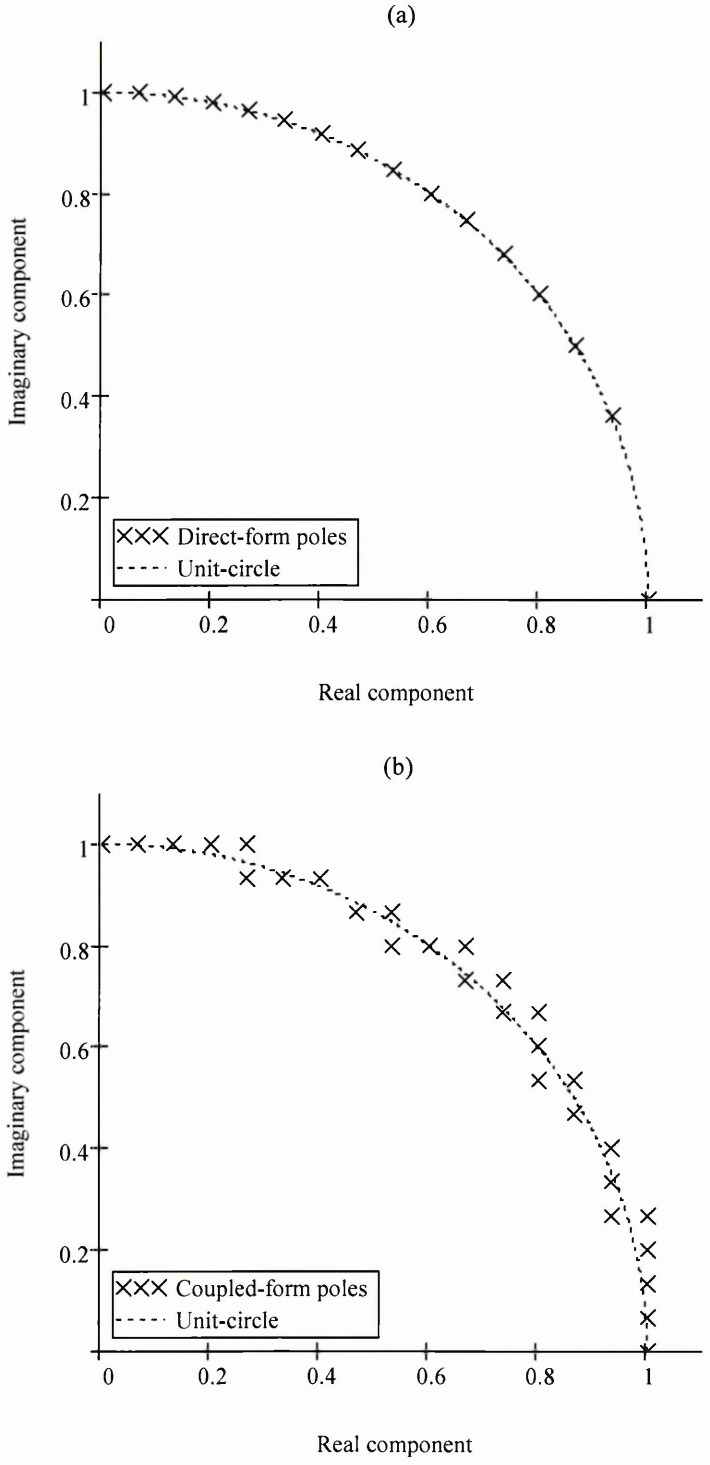


Figure (3.2.5): Pole distribution around the first quadrant of the unit-circle for the direct-form (a) and coupled-form (b) recursive oscillators with quantised arithmetic. (Oscillator coefficients are quantised to 16 levels on the interval $[0,1]$ *to exaggerate pole distribution effects.)*

Figure (3.2.5b) illustrates two important results for the coupled-form oscillator – the pole locations do not always lie exactly on the unit circle ($r \neq 1$), but are distributed *uniformly* around it (i.e. the angular separation between adjacent poles is constant). We define frequency *resolution* as the frequency change corresponding to a unit change in the quantised frequency control coefficient. Uniform pole distribution as exhibited by the coupled-form oscillator provides improved (i.e. decreasing) frequency resolution at low frequencies in contrast to the direct-form oscillator whose *non-uniform* pole distribution is illustrated in Figure (3.2.5a) [Oppenheim & Schafer, 1975]. The direct-form pole distribution exhibits increasing angular separation between adjacent poles as frequency tends to zero. Since frequency resolution can be equated to the minimum possible angular separation between adjacent poles, we observe it is not constant and increases with reducing oscillation frequency. Figure (3.2.6) illustrates the angular position of low frequency pole-pairs for the direct-form oscillator and the relationship with absolute frequency and frequency resolution, f_r.

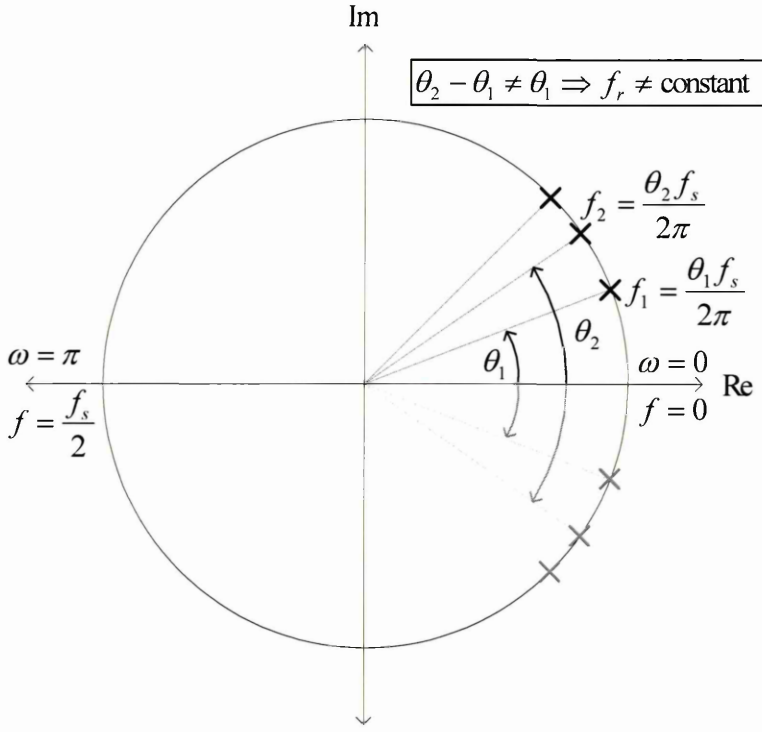


Figure (3.2.6): Pole distribution around the unit-circle in the complex z-plane illustrating the non-uniform distribution of low frequency conjugate pole pairs for the direct-form oscillator.

Amplitude errors due to pole deviation from the unit circle can be reduced by reinitialising the oscillator at periodic intervals when the amplitude error has exceeded a predefined threshold [Curticapean *et al*, 2000]. The question of how to determine the re-initialisation period can be addressed by considering the amplitude envelope, r^n. The quantised pole radius, r_q, with quantised frequency, θ_q, is given by $r_q = \sin^2(\theta_q) + \cos^2(\theta_q)$. We next define the pole unit-radius deviation error, d, where $d = |(r_q - 1)|$. For very small d, which applies for most practical quantisation intervals, we observe that the amplitude growth and decay envelopes ($(1+d)^n$ and $(1-d)^n$, respectively) are mutually reciprocal, that is $(1+d)^n \cong (1-d)^{-n}$. Therefore the metric of $\max(d)$ computed over the operating frequency interval can be used to determine a

worst case amplitude envelope, $(1 \pm d)^n$, for a particular quantisation interval. The amplitude envelope can be used to determine the re-initialisation period for a particular amplitude error. Figure (3.2.7) plots the number of elapsed samples (i.e. time) following initialisation for a given oscillation amplitude change against word size, b, assuming a fixed-point number representation which gives a quantisation interval of $2^{-(b-1)}$. The amplitude change is computed using the value of max(d) over the Nyquist interval of 24 kHz and therefore represents the *worst case* amplitude error for a given arithmetic quantisation.

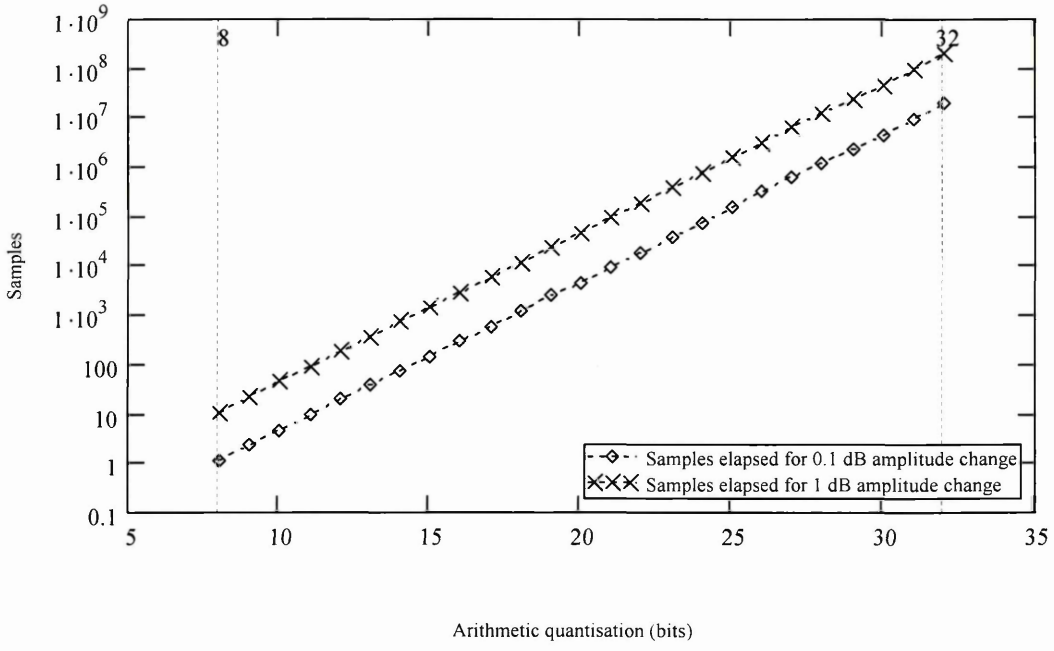


Figure (3.2.7): Elapsed samples for a given oscillation amplitude change against word size varying from 8 to 32 bits assuming a fixed-point number representation computed over the Nyquist interval of 24 kHz.

The coupled-form oscillator permits sample rate frequency control without exhibiting phase discontinuities typical of the direct-form. Frequency transition is *inherently* phase-continuous and does not require computation of new IC values. The oscillator requires four multiplies and two additions per sample, with further computational

overhead associated with re-initialisation to correct amplitude deviation over n. Dynamic linear frequency control requires sample rate computation of $\cos(\theta(n))$ and $\sin(\theta(n))$, with $\theta(n) = 2\pi F(n)T$. Amplitude control requires an additional multiply operation, or two if a complex (quadrature) output is required.

3.2.3 Modified Coupled-form

An improved coupled-form algorithm overcoming the inherent quantisation sensitivity and computation cost has been suggested by Gordon and Smith [1985]. The so-called modified coupled-form oscillator is illustrated in Figure (3.2.8) and is characterised by the vector multiplication:

$$\begin{bmatrix} y_1(n) \\ y_2(n) \end{bmatrix} = \begin{bmatrix} 1 & -\varepsilon \\ \varepsilon & (1-\varepsilon^2) \end{bmatrix} \begin{bmatrix} y_1(n-1) \\ y_2(n-1) \end{bmatrix} \qquad (3.2.10)$$

where $\varepsilon = 2\sin\left(\frac{\theta}{2}\right)$.

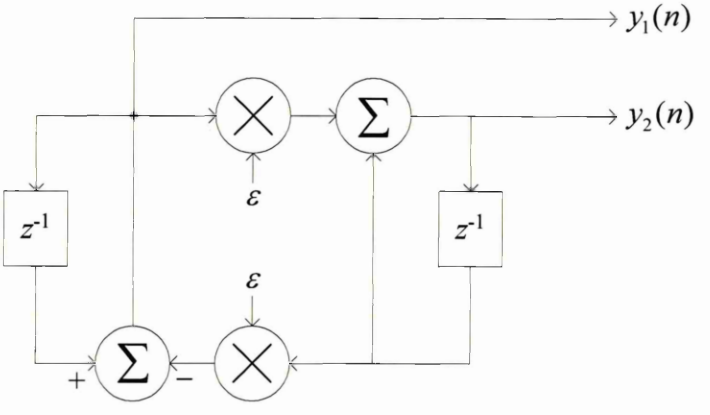


Figure (3.2.8): The modified coupled-form recursive oscillator.

The algorithm requires only two multiplies per sample, is first order and most significantly does *not* suffer from the quantisation sensitivities associated with maintaining $r = 1$ as seen with the coupled-form oscillator. The matrix determinant represents the vector scaling (r) and is unity for *all* values of ε and therefore

independent of quantisation effects. The frequency may be dynamically updated at the sample rate requiring computation of ε for each new frequency, and produces phase-continuous frequency transitions.

Gordon and Smith [1995] present an alternative form of Eqs. (3.2.10) which considers the phase relationship between $y_1(n)$ and $y_2(n)$. We have:

$$\begin{bmatrix} y_1(n) \\ y_2(n) \end{bmatrix} = \mathbf{G} \begin{bmatrix} y_1(-1) \\ y_2(-1) \end{bmatrix} \tag{3.2.11}$$

where $\mathbf{G} = \frac{1}{\sin(\varphi)} \begin{bmatrix} \sin(n\theta + \varphi) & -\sin(n\theta) \\ \sin(n\theta) & -\sin(n\theta - \varphi) \end{bmatrix}$ and $\varphi = \frac{(\pi - \theta)}{2}$. If we set $y_1(-1) = 1$ and $y_2(-1) = \cos(\varphi)$ we obtain $y_1(n) = \cos(n\theta)$ and $y_2(n) = \sin(n\theta - \varphi)$. As θ approaches zero, the phase between $y_1(n)$ and $y_2(n)$ approaches $\frac{\pi}{2}$ (quadrature). However, as the magnitude of θ increases and approaches $\pm\pi$, the phase between $y_1(n)$ and $y_2(n)$ approaches zero (in phase). The modified coupled-form oscillator does not provide frequency independent quadrature between $y_1(n)$ and $y_2(n)$.

3.2.4 Waveguide-form

The second order digital waveguide oscillator, derived from digital waveguide theory, has been proposed by Smith and Cook [1992] and is illustrated in Figure (3.2.9).

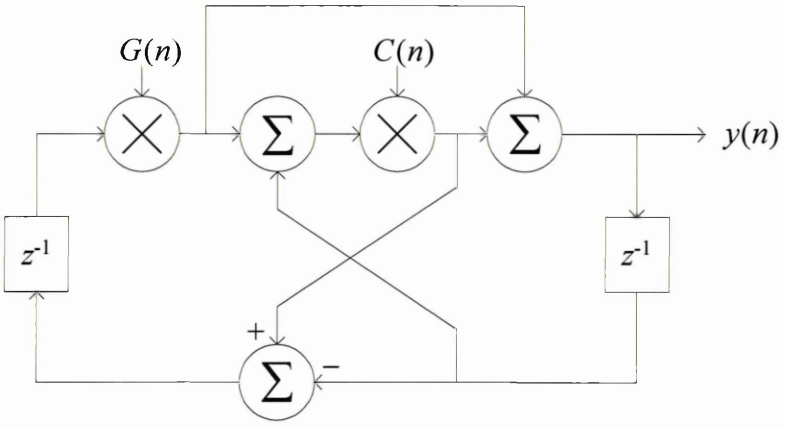


Figure (3.2.9): The waveguide-form recursive oscillator.

This form requires one multiply and three additions per sample when amplitude and frequency are constant. Frequency transitions are intrinsically phase-continuous and do not require computation of new IC values. However, an additional multiply operation is needed at each frequency transition to normalise the amplitude. In contrast to the coupled-form, the waveguide-form oscillator does *not* suffer exponential amplitude drift due to quantisation round-off errors since rounding occurs *only* at the tuning multiplication involving $C(n)$ and all other computations are exact. Quantisation in the tuning coefficient, $C(n)$, can only cause quantisation in the frequency of oscillation [Smith and Cook, 1992]. We have:

$$C(n) = \cos(\theta(n))$$

$$g(n) = \sqrt{\frac{(1-C(n))}{(1+C(n))}} \qquad (3.2.12)$$

$$G(n) = \frac{r(n)g(n)}{g(n-1)}$$

where $\theta(n) = 2\pi F(n)T$ and $r(n)$ is the exponential growth or decay per sample, with $r(n) = 1$ for constant amplitude. When both amplitude and frequency are constant, we

have $G(n)=1$ and only the tuning multiply is required. Upon a frequency transition, $G(n)$ deviates from unity for one sample to normalise the amplitude. The normalisation coefficient, $G(n)$, incurs considerable computation overhead. For $r(n)=1$ and a frequency transition at sample m, we have $G(m)=\sqrt{\frac{(1-C(m))}{(1+C(m))}\frac{(1+C(m-1))}{(1-C(m-1))}}$.

The authors report that the waveguide-form is suitable for VLSI implementation and can be readily applied to recursive FM synthesis. The waveguide-form offers little improvement on the direct-form due to the computation overhead associated with computing $G(n)$ at each frequency transition, particularly with sample rate control of $F(n)$.

3.2.5 Frequency Control and Quantisation Effects

All recursive algorithms exhibit adverse behaviour with quantised samples and frequency control coefficient(s). If we assume that the coefficient(s) and signal samples are represented by b fractional bits in a fixed-point number representation, two quantisation effects are evident – frequency control sensitivity and computation round-off errors. Coefficient quantisation displaces the poles from their intended (desired) positions on the unit circle. If the poles do not lie on the unit circle then $r \neq 1$ and a sinusoid with exponentially decaying or growing amplitude is produced. If the poles are located incorrectly on the unit circle due to quantisation effects, a constant amplitude sinusoid is generated but with a frequency different from that intended. All *single* multiplier oscillators have poles located precisely on the unit circle but at different locations from the ideal (non-quantised) case [Abu-El-Haija *et al*, 1986]. The direct-form oscillator has poles which lie precisely on the unit circle but are not uniformly distributed around it (as depicted in Figure (3.2.5a)). For a given quantisation interval,

we observe increased pole separation at low frequencies and a corresponding reduction in frequency control resolution. For the direct-form coefficient, $2\cos(\theta)$, θ can only take on a finite number of values to ensure $2\cos(\theta)2^b$ takes on integer values. Furuno *et al* [1975] report that the frequency coefficient must be implemented in terms of $\hat{\theta}$ with $\hat{\theta} < \theta$ and $2\cos(\hat{\theta}) = 2^{-b}\lfloor 2\cos(\theta)2^b + 1 \rfloor$. The actual oscillation frequency is then less than the desired frequency.

Each coefficient multiplication produces a $2b$ bit product which must be truncated or rounded to b bits on each recursion. Rounding is preferred to truncation since it makes some use of the discarded information, but essentially the least significant b bits of information are lost on each multiplication. (The nature of 2's complement coding causes the direct-form -1 multiplication to fit precisely within b-bits and so is absolutely precise.) Addition or subtraction of quantised samples produces results which fit within the operand word size provided arithmetic overflow or underflow is prevented.

All recursive oscillators reported in the literature have control coefficients which are a sine or cosine function of the oscillation frequency (θ). None provide a linear transfer function, which is a desirable property for musical additive synthesis. A linear relationship between oscillation frequency and $F(n)$ therefore requires computation of the particular trigonometric coefficient equation in all cases.

Quantisation causes the oscillation frequency to differ from the limiting case (full arithmetic precision), with error magnitude depending on the quantisation interval and the coefficient equation. In musical applications we are concerned with *relative* accuracy, that is, how precisely we can represent the *ratio* of two frequencies. This arises from the fixed ratio of $\sqrt[12]{2}$ between adjacent semitone frequencies in the equally

tempered musical scale and a unit of measure in this regard is the *cent*, defined as $\frac{1}{100}$ of a semitone or a frequency ratio of $\sqrt[1200]{2}$ [Chamberlin, 1985]. In particular, we require maximum frequency control resolution at the lowest frequencies when the semitone frequency difference is small. For example, a one semitone shift at 27.5 Hz (A0) is approximately 1.64 Hz. Conversely, a one semitone shift at 440 Hz (A4) is approximately 26.16 Hz. Coefficients which are a *cosine* function of θ (i.e. the direct and waveguide-forms) exhibit reduced frequency resolution at low frequencies with fixed-point arithmetic. As θ tends to zero the slope of the coefficient $2\cos(\theta)$ also tends to zero and therefore progressively more bits are required to represent $2\cos(\theta)$ to a given accuracy.

Hodes *et al* [1999] present a quasi floating-point direct-form algorithm which provides greatly improved frequency resolution at low frequencies but requires two additional operations per sample – an add with fixed shift and a variable barrel shift which increases computational overhead considerably.

Coefficients which are a *sine* function of θ (i.e. the modified coupled-form) exhibit increased frequency resolution at low frequencies for a given quantisation interval compared to the direct and waveguide-forms. Figure (3.2.10a) illustrates the frequency control characteristics for the direct-form and modified coupled-form oscillators with quantisation interval sufficiently large to expose these effects ($b = 5$ and $f_s = 48$ kHz).

For the direct and waveguide-forms, the ratio between two frequencies separated by one least significant bit with a b-bit arithmetic quantisation is given by:

$$\varepsilon_1 = \frac{\cos^{-1}\left(\frac{\lfloor 2\cos(\theta)2^b \rfloor}{2^{b+1}}\right)}{\cos^{-1}\left(\frac{\lfloor 2\cos(\theta)2^b \rfloor + 1}{2^{b+1}}\right)} \qquad (3.2.13)$$

Eq. (3.2.13) defines a relative tuning error metric which we plot in Figure (3.2.10b) over frequency for three quantisation values. This ratio can be considered as a relative tuning error for a particular quantisation interval. Moore [1990] suggests that the smallest frequency ratio distinguishable by humans is around 5 cents (i.e. $2^{\frac{5}{1200}}$ or approximately 1.0029) assuming an equally tempered scale. It is evident from Figure (3.2.10b) that tuning error for low frequencies is above 5 cents with less than 24 bit arithmetic, becoming progressively worse for 20 and 16 bit arithmetic.

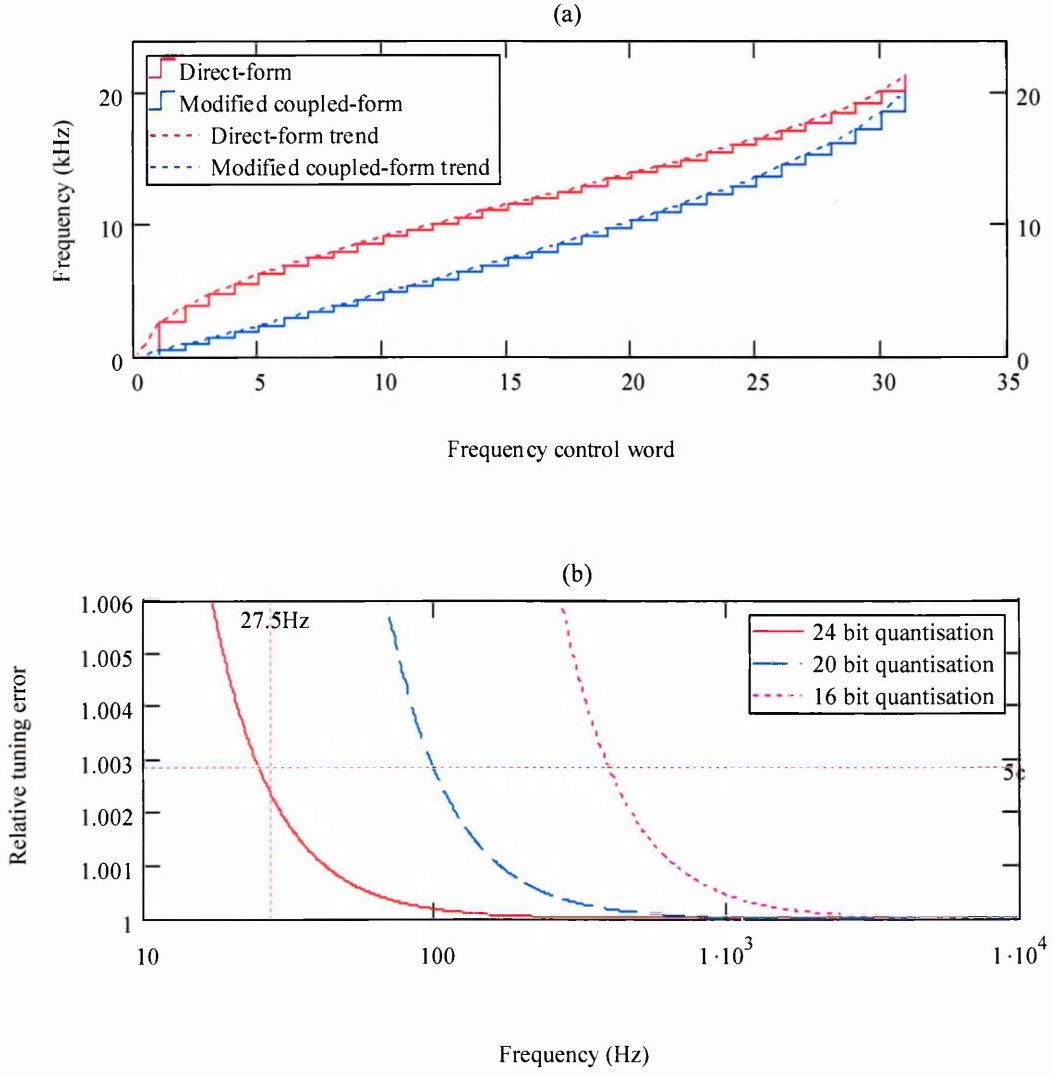


Figure (3.2.10): (a) – Quantised frequency control characteristics for the direct-form and modified coupled-form oscillators. (b) – Relative tuning error for the direct and waveguide-form oscillators with b = 24, 20 and 16 bits.

The coupled-form exhibits a uniform pole distribution over the whole Nyquist interval and therefore a constant frequency resolution irrespective of absolute frequency. Coefficient computation using a scaled $F(n)$ argument represents a significant computational overhead – the scaling multiplication of $F(n)$ by $2\pi T$ and the subsequent sine or cosine operation. The trigonometric mapping must be performed with sufficient precision to ensure adequate $F(n)$ resolution. This overhead is common to all recursive oscillators discussed in the literature.

Computation round-off error leads to an accumulative error in $y(n)$ whose nature depends on the particular algorithm. Round-off errors arise because the output of the coefficient multiplier must be quantised to b bits at every iteration causing the oscillator output to deviate from the ideal over time. We can consider round-off error as the variance of the output noise caused by post-multiplication quantisation, similar to that presented in the recursive digital filter literature. The direct-form oscillator round-off error variance over N_s samples is approximated by $\sigma^2(N_s) \cong \frac{2^{-2b} N_s}{6\sin^2(\hat{\theta})}$, when $N_s >> \frac{2\pi}{\hat{\theta}}$ [Abu-El-Haija *et al*, 1986]. The noise variance (i.e. noise power) increases steadily with N_s and is inversely proportional to $\sin^2(\hat{\theta})$. The direct-form oscillator therefore requires periodic re-initialisation to prevent the build up of excessive round-off noise and hence signal-to-noise ratio (SNR) falling below an acceptable level. The coupled-form oscillator exhibits only amplitude error due to quantisation effects, however, re-initialisation is still necessary in practice. The modified coupled-form provides invariant oscillation amplitude with coefficient quantisation, with SNR bound only by sample quantisation noise.

3.2.6 Initial Conditions and Phase Continuity

The coupled-form, modified coupled-form and waveguide-form recursive oscillators produce phase-continuous, constant amplitude sinusoids following a transition in the frequency control parameter. The direct-form oscillator is attractive due to its low computational overhead [Hodes *et al*, 1999], but requires computation of new ICs at *every* frequency transition to maintain phase-continuity. We therefore investigate the computational overhead associated with the direct-form IC values required for constant amplitude, phase-continuous frequency transition.

To effect a constant-amplitude, phase-continuous frequency transition from ω to ω' at sample index, m, using ICs from Eq. (3.2.6), we require the phase of the sinusoid at sample index $m-1$, just before the frequency change at m. We define the phase of a particular sample with respect to the most recent zero phase point (cycle start) in the sinusoidal sequence and within the interval $[0, 2\pi)$. We may modulo-2π accumulate the oscillator *phase increment*, ωT, synchronous with the recursive oscillator process to compute the oscillation phase at a particular sample index. The phase is given by $\phi(n) = \langle n\omega T + \phi_0 \rangle_{2\pi}$, where ϕ_0 represents the initial phase. The modulo-2π operation is achieved by using an M-bit accumulator performing unsigned integer arithmetic with initial condition $\left\lfloor \frac{\phi_0}{2\pi} 2^M \right\rfloor$ and $\phi_0 \in [0, 2\pi)$. $\left\lfloor \frac{\omega T}{2\pi} 2^M \right\rfloor$ is accumulated modulo-2^M over n producing the output $\left\langle \left\lfloor \frac{n\omega T + \phi_0}{2\pi} 2^M \right\rfloor \right\rangle_{2^M}$ after n samples. Multiplying the accumulator output by $\frac{2\pi}{2^M}$ effects modulo-2π scaling. This approach requires two multiplication operations and an M-bit accumulator, with M chosen to give the desired phase resolution of $\frac{2\pi}{2^M}$ radians. This technique provides optimal prediction of

the oscillation phase at sample n to a resolution governed by M, but will not precisely track the phase of a recursively generated sinusoid due to round-off errors with fixed-point arithmetic. Figure (3.2.11) illustrates simulated phase error behaviour with sample index for the direct-form oscillator computed with fixed-point arithmetic. (The phase of the recursively generated sinusoid has been computed using a full precision floating point arcsine function.)

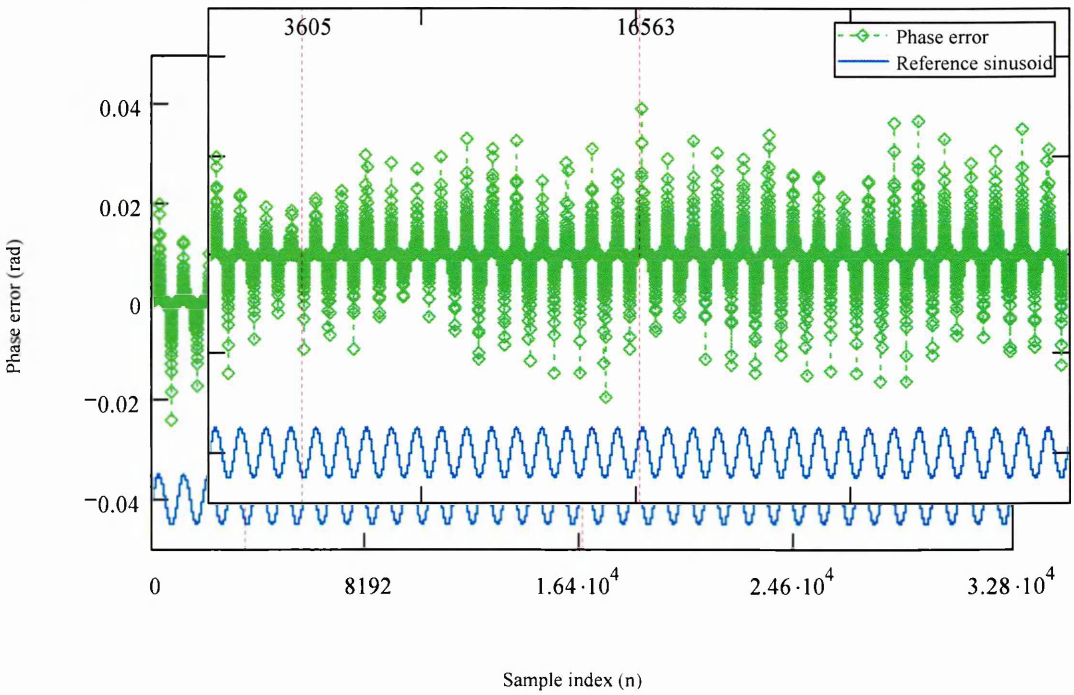


Figure (3.2.11): Simulated phase error between phase accumulator and the direct-form oscillator with $f_s = 48$ kHz, $f = 50$ Hz and 24 bit fixed-point arithmetic. An offset reference sinusoid is also shown and indicates that maximum error occurs at the turning points of $y(n)$ as exemplified by the two markers at $n = 3605$ and $n = 16563$.

If we represent the DT phase argument, $(n\theta + \phi)$, in Eq. (3.2.5) as $\Phi(n)$, and assume a unit amplitude sinusoid ($A = 1$), then $y(n) = \sin(\Phi(n))$ and so $\Phi(n) = \sin^{-1}(y(n))$. However, the trigonometric functions are not one-one on their whole domains. To obtain inverse functions, each trigonometric function is restricted to a subset of the

domain where it is one-one. The sine function is one-one on the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$ with range $[-1, 1]$. The inverse sine function thus maps $[-1,1] \rightarrow [-\frac{\pi}{2}, \frac{\pi}{2}]$ in a one-one manner. To unambiguously determine the phase of a DT sinusoid sample there are four quadrants to consider with phase intervals $[0, \frac{\pi}{2})$, $[\frac{\pi}{2}, \pi)$, $[\pi, \frac{3\pi}{2})$ and $[\frac{3\pi}{2}, 2\pi)$ respectively. Considering the sign of $y(n)$ and the slope at $y(n)$ allows the interval containing $y(n)$ to be determined. For a frequency change at $y(m)$, the slope, λ, at $y(m-1)$ is *approximately* proportional to $y(m-1)-y(m-2)$. Eq. (3.2.14) gives the end-point phase, $\phi(m-1)$, taking account of the particular phase-interval (quadrant) in which $\phi(m-1)$ lies, where $\wedge$ denotes the Boolean AND operator:

$$\phi(m-1) = \begin{cases} \sin^{-1}(y(m-1)), & y(m-1) > 0 \wedge \lambda > 0 \\ \pi - \sin^{-1}(y(m-1)), & \lambda < 0 \\ 2\pi + \sin^{-1}(y(m-1)), & y(m-1) < 0 \wedge \lambda > 0 \end{cases} \qquad (3.2.14)$$

where $\lambda = y(m-1)-y(m-2)$. Eq. (3.2.14) defines $\phi(m-1)$ across all four phase quadrants of the sinusoid cycle. (Substituting $n+1$ for m in Eq. (3.2.14) gives a general expression for $\phi(n)$ as a function of $y(n)$ and $y(n-1)$.)

We now present a method for obtaining a constant amplitude, phase-continuous, transition from frequency ω to ω'. Eq. (3.2.14) gives the DT sinusoid phase at the last sample of the 'first' recursion, $m-1$. The phase value, ϕ, used in Eq. (3.2.6) is given by:

$$\phi = \phi(m-1) + \theta' \qquad (3.2.15)$$

where the θ' term represents the phase increment from the last sample at the original frequency, ω, to the phase start point of the new recursion at frequency ω', with $\theta' = 2\pi\omega'$. When substituting ϕ from Eq. (3.2.15) into Eq. (3.2.6) to compute $y'(-1)$

and $y'(-2)$ for the recursion at frequency ω', we consider two cases depending on the phase interval containing $\phi(m-1)$. We have $y'(-1)=\frac{A'}{A}y(m-1)$, where A and A' are the amplitudes of the original and new recursions respectively, which we include for generality. We also observe the special case $y'(-1)=y(m-1)$ for unit amplitude sinusoids or $A=A'$.

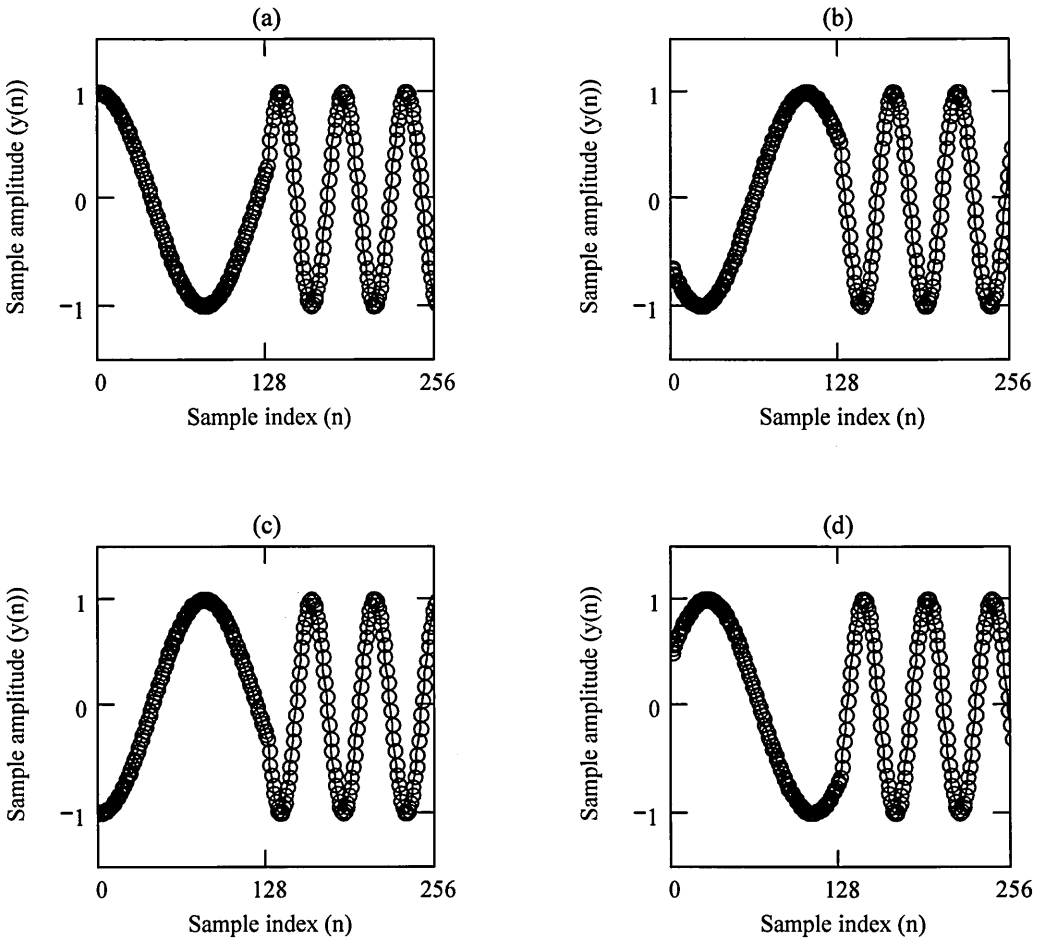
The IC $y'(-2)$ takes one of two values depending on the slope of $y(n)$ just before the frequency change. Hence:

$$y'(-1)=\frac{A'}{A}y(m-1)$$

$$y'(-2)=\begin{cases} A'\cos\left(\cos^{-1}\left(\frac{y(m-1)}{A}\right)+\theta'\right), & \lambda>0 \\ A'\cos\left(\cos^{-1}\left(\frac{y(m-1)}{A}\right)-\theta'\right), & \lambda<0 \end{cases} \tag{3.2.16}$$

where $\lambda=\frac{y(m-1)-y(m-2)}{A}$.

Figures (3.2.12a) through (3.2.12d) illustrate constant amplitude, phase continuous frequency transitions using ICs obtained from Eqs. (3.2.16), and are arranged to occur in each quadrant of the $y(n)$ sequence. The technique is seen to give good results with a discontinuity-free transition between the two frequencies.



Figures (3.2.12a) through (3.2.12d): Constant amplitude, phase continuous frequency transition in each phase quadrant using ICs obtained from Eqs. (3.2.15). Frequency transition occurs at $n=128$ *with the initial phase of the first recursion chosen to position the transition point in each quadrant.*

The ICs given by Eqs. (3.2.15) give good results for oversampled signals, that is for frequencies $\omega << \frac{2\pi}{T}$. When this condition is met, the ICs will provide a constant amplitude, phase-continuous frequency change in any of the four quadrants of the $y(n)$ sequence. We note that the formulation of Eqs. (3.2.16) is more comprehensive than that proposed by Lane *et al* [1997]. Using the notation of this thesis, the latter takes the form $y'(-1) = y(m-1)$ and $y'(-2) = \cos\left(\cos^{-1}[y(m-1)] - \theta'\right)$.

The single expression for $y'(-2)$ reflects that these ICs are valid only for unit amplitude sinusoids and frequency transitions occurring in the interval $[\frac{\pi}{2}, \frac{3\pi}{2}]$ where the slope of $y(n)$ is negative. Frequency transitions occurring when the slope of $y(n)$ is positive produce phase discontinuities between $y(n)$ and $y'(n)$ using this method.

Eqs. (3.2.16) use the slope of the line between $y(m-1)$ and $y(m-2)$ to approximate the instantaneous derivative (slope) at $y(m-1)$ and thereby give the particular phase interval containing $y(m-1)$. (i.e. $\lambda > 0 \Rightarrow \phi(m-1) \in [0, \frac{\pi}{2}) \cup [\frac{3\pi}{2}, 2\pi)$ and $\lambda < 0 \Rightarrow \phi(m-1) \in [\frac{\pi}{2}, \frac{3\pi}{2})$.) On closer inspection we find that this method produces an incorrect determination of the phase interval containing $y(m-1)$ near local maximum or minimum points and incorrectly places $y(m-1)$ in the next *lower* phase quadrant under certain conditions. The phase interval over which this error can occur is governed by the phase increment, ωT, and approaches a limiting value dictated by the Nyquist sampling criterion. As ωT approaches the Nyquist limit value of π, the width of the error interval approaches a maximum value of $\frac{\pi}{2}$ radians. This phase error causes a phase discontinuity between $y(n)$ and $y'(n)$ if the frequency transition occurs near the turning points of $y(n)$. The magnitude of the phase discontinuity increases as ωT approaches π. Eqs. (3.2.16) yield a progressively more accurate $y'(-2)$ value in the vicinity of a turning point in $y(n)$ as ωT and hence the width of the error interval tends to zero. In general, the width of the error interval at a particular frequency, ω, is $\frac{\omega T}{2}$ radians. Figure (3.2.13) shows a contour plot where the vertical axis represents the start phase of the first recursion and the horizontal axis represents sample index with

frequency transition occurring at $n = 50$. Contour lines represent lines of constant amplitude in $y(n)$. As the start phase varies, the fixed transition sample index causes the frequency transition phase to span 2π radians. Transition anomalies are seen in the second recursion (right hand side of Figure (3.2.13)) for particular start phase values which correspond to a transition point near the turning points in $y(n)$.

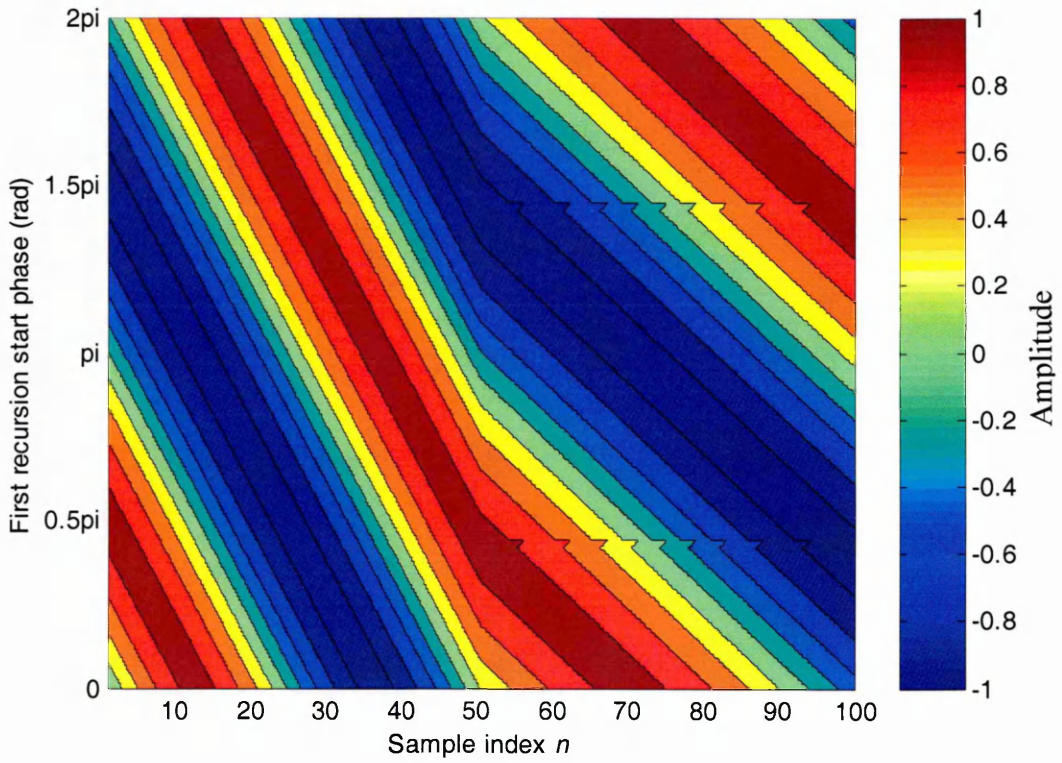
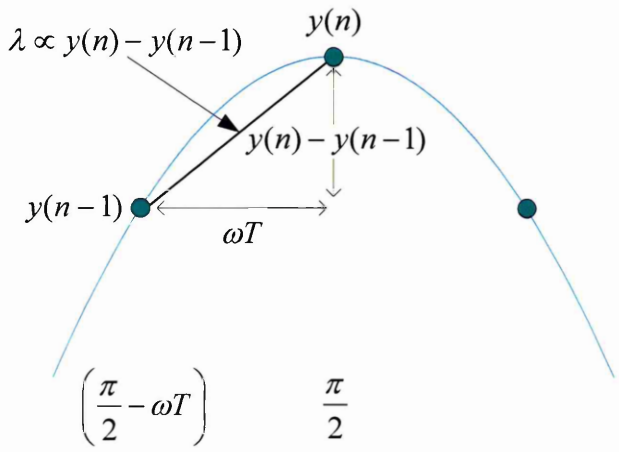


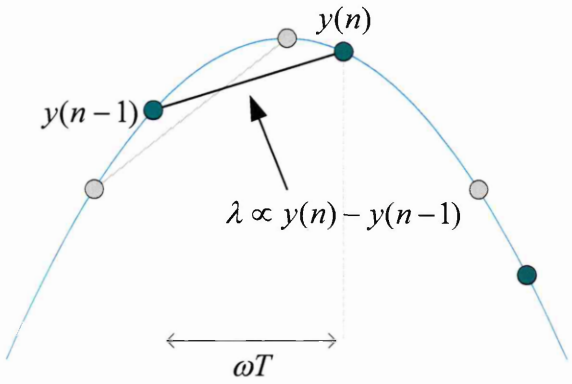
Figure (3.2.13): Contour plot illustrating the performance of Eqs. (3.2.16). The vertical axis represents start phase and the horizontal axis represents sample index. Contour lines depict lines of constant amplitude in $y(n)$. Transition occurs at $n = 50$, with $f = 2\pi\omega = 1000$ Hz, $f' = 2\pi\omega' = 500$ Hz and $f_s = 48$ kHz. (Notice the discontinuities in the right hand region of the plot for certain transition phases.)

Figures (3.2.14) parts (a), (b) and (c) provide a graphic illustration of this error mechanism for the positive turning point of a generalised unit-amplitude DT sine sequence, $y(n)$, of frequency ω and sampling period T. The slope, λ, of the line

between two consecutive samples, $y(n)$ and $y(n-1)$, is $\frac{y(n)-y(n-1)}{\omega T}$ which we normalise to $y(n)-y(n-1)$ for given signal and sampling frequencies. Eqs. (3.2.16) test the sign of λ to determine the slope on which $y(n)$ is located. Figure (3.2.14a) shows the limiting case for the $\lambda > 0$ condition, placing $y(n)$ correctly in the first quadrant. λ reaches a limiting value of λ_m when $y(n)$ is located *exactly* at the maximum point on the sine curve. The magnitude of λ_m is therefore $1-\sin\left(\frac{\pi}{2}-\omega T\right)=1-\cos(\omega T)$, assuming a unit amplitude sinusoid, and the condition holds for both turning points by symmetry. The condition $|\lambda| > \lambda_m$ places $y(n)$ in quadrant 1 or 4, that is, on the positive slope of the sine curve. λ values satisfying $0 < \lambda < \lambda_m$ can still occur over an interval when $y(n)$ is in the second quadrant as shown in Figure (3.2.14b). In this region the $\lambda > 0$ condition would conclude that $y(n)$ is in quadrant 1 when it is actually in quadrant 2 – a region of negative sine slope. This condition is maintained until $\lambda = 0$ which allows the error interval to be determined by geometric inspection as depicted in Figure (3.2.14c). A similar argument applies for the negative turning point between quadrants 3 and 4 due to the symmetry of the sine function.



(a)



(b)

Figures (3.2.14a) and (3.2.14b): Discriminating the phase of $y(n)$ between quadrants 1 and 2 by examining the slope of the line between $y(n)$ and $y(n-1)$.

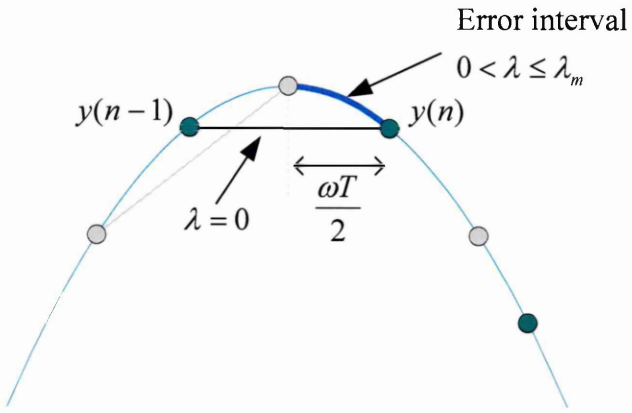


Figure (3.2.14c): The error interval where $y(n)$ can be placed in the incorrect quadrant. ($y(n)$ is clearly in quadrant 2 but would be determined to be in quadrant 1 over the error interval when using the $\lambda > 0$ condition as in Eqs. (3.2.15).)

By applying test conditions in line with considerations from Figure (3.2.14c) to λ and $y(n)$, we proceed to define an expression which gives the phase of a DT sinusoid with amplitude A at sample $y(n)$ given only $y(n)$ and $y(n-1)$, thus:

$$\phi(n) = \begin{cases} \sin^{-1}\left(\frac{y(n)}{A}\right), & y(n) \ge 0 \wedge \lambda > \lambda_m \\ \pi - \sin^{-1}\left(\frac{y(n)}{A}\right), & (y(n) \ge 0 \wedge \lambda \le \lambda_m) \vee (y(n) < 0 \wedge -\lambda > \lambda_m) \\ 2\pi + \sin^{-1}\left(\frac{y(n)}{A}\right), & y(n) < 0 \wedge -\lambda \le \lambda_m \end{cases} \qquad (3.2.17)$$

where $\lambda = \frac{y(n) - y(n-1)}{A}$ and $\lambda_m = 1 - \cos(\omega T)$. The test condition intervals of Eq. (3.2.17) are illustrated in Figure (3.2.15).

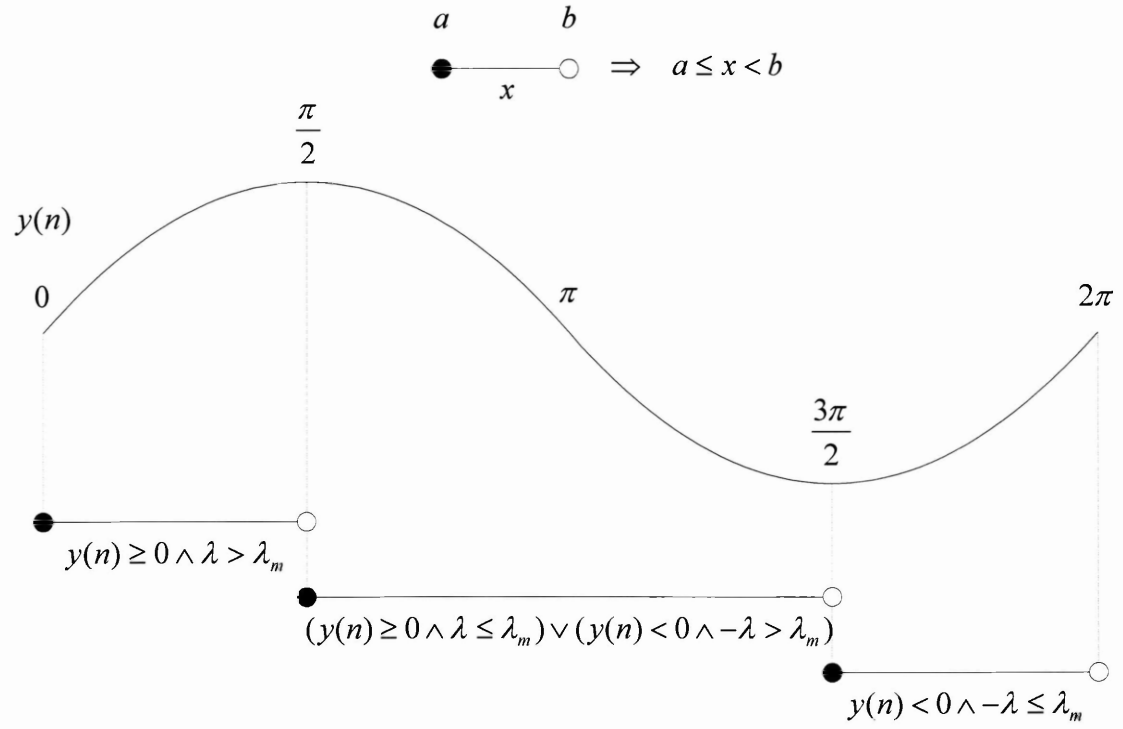


Figure (3.2.15): Phase intervals corresponding to the test conditions in Eq. (3.2.17), illustrating their position across a single sinusoid cycle.

Using Eq. (3.2.17) we therefore define optimal values for the ICs $y'(-1)$ and $y'(-2)$ in terms of $y(m-1)$, $y(m-2)$, A, A', and ω', thus:

$$y'(-1) = \frac{A'}{A} y(m-1)$$

$$y'(-2) = \begin{cases} A' \sin\left(\sin^{-1}\left(\dfrac{y(m-1)}{A} \right) - \theta' \right), & (y(m-1) \geq 0 \wedge \lambda > \lambda_m) \vee (y(m-1) < 0 \wedge -\lambda \leq \lambda_m) \\ A' \sin\left(\sin^{-1}\left(\dfrac{y(m-1)}{A} \right) + \theta' \right), & (y(m-1) \geq 0 \wedge \lambda \leq \lambda_m) \vee (y(m-1) < 0 \wedge -\lambda > \lambda_m) \end{cases}$$

(3.2.18)

where $\theta' = \omega' T$, $\lambda = \dfrac{y(m-1) - y(m-2)}{A}$ and $\lambda_m = 1 - \cos(\omega T)$.

For a constant amplitude frequency change (i.e. $A' = A$), the initial conditions given by Eqs. (3.2.18) produce a precisely phase-continuous transition. Figure (3.2.16) illustrates

a contour plot showing constant amplitude, phase continuous frequency transitions over a 2π range of start phase values using Eqs. (3.2.18) to generate the ICs for the new recursion. There are no phase discontinuities evident in contrast to Figure (3.2.13).

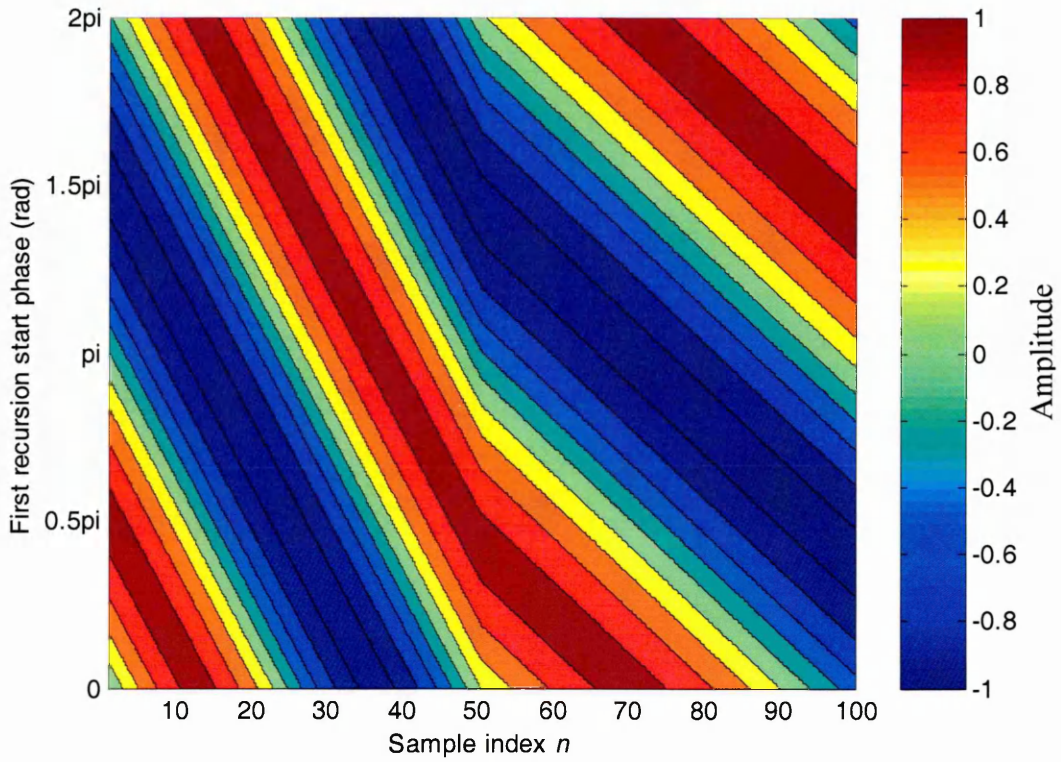


Figure (3.2.16): Contour plot illustrating the performance of Eqs. (3.2.18). The vertical axis represents phase and the horizontal axis represents sample index. Contour lines depict lines of constant amplitude in $y(n)$. *Transition occurs at* $n = 50$, *with* $f = 2\pi\omega = 1000$ *Hz,* $f' = 2\pi\omega' = 500$ *Hz and* $f_s = 48$ *kHz. (Notice the absence of horizontal discontinuities in the right hand region of the plot.)*

Sample quantisation causes Eqs. (3.2.17) and (3.2.18) to produce erroneous results under particular conditions. The error magnitude can be reduced by increasing sample word length and thereby reducing the quantisation interval. A unit-amplitude, 2's complement fixed-point fractional number representation of b bits has a range interval of $[1-2^{-(b-1)}, -1]$ with quantisation interval $q = 2^{-(b-1)}$. Assuming a unit-amplitude DT

sinusoid with quantisation interval, q, and sample interval, T, there will be a frequency, $\frac{\sin^{-1}(q)}{2\pi T}$, *below* which the sinusoid, $y(n)$, is sufficiently *oversampled* as to cause groups of adjacent samples to lie within the *same* quantisation interval across the range $[1-2^{-(b-1)}, -1]$. This leads to $y(n)-y(n-1)=0$ even though the slope of the underlying sinusoid function is non-zero and causes erroneous behaviour of Eqs (3.2.17) and (3.2.18). As frequency increases above $\frac{\sin^{-1}(q)}{2\pi T}$ the region where adjacent samples lie within the same quantisation interval moves away from the zero crossing of $y(n)$ (where the slope is a maximum) toward the turning point. Residual errors remain in the vicinity of a turning point in $y(n)$, reducing only with an increased number of bits to represent the sample. Behaviour of the *maximum* phase error measured over one cycle with b is shown in Figure (3.2.17) for a frequency of 32 Hz and fixed-point quantisation.

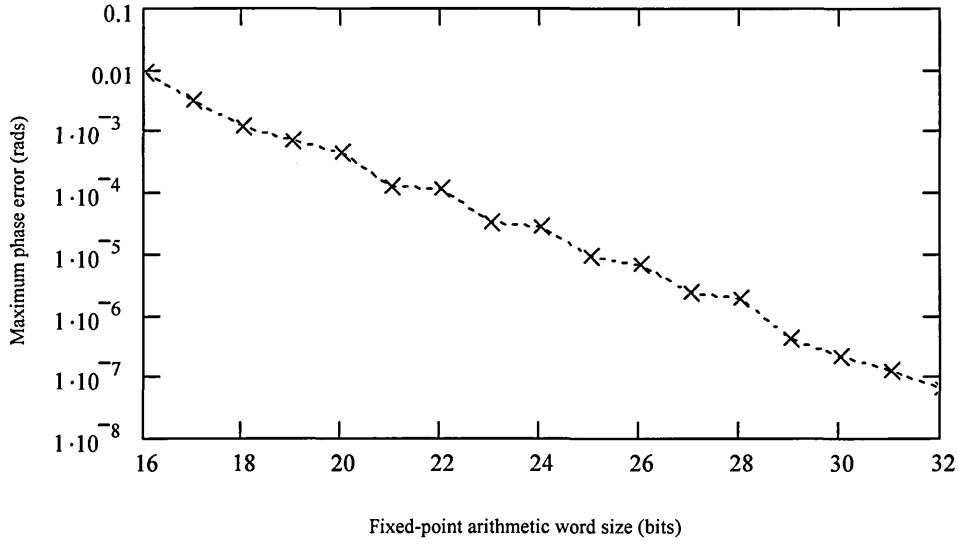


Figure (3.2.17): Variation of peak phase error with quantisation bits for Eq. (3.2.17) with $f = 32$ Hz and $f_s = 48$ kHz. (A low frequency test signal is used since error magnitude increases with reducing frequency.)

3.3 Phase Accumulating Sinusoidal Oscillators

The concept of phase-accumulating frequency synthesis traces its origins to the pioneering work of Mathews [1969] in connection with computer music signal generation and Tierney *et al* [1971] in connection with generalised frequency synthesis. The phase-accumulating oscillator exploits the property of an overflowing M-bit accumulator to generate a modulo-2^M sequence used as an argument to a function which maps from the phase to amplitude domain. Phase accumulating frequency synthesis is a "ground up" technique involving essentially two stages – phase sequence generation and phase-amplitude mapping. We review phase-accumulation as a precursor to the more general *wavetable lookup* synthesis in Chapter 4. In this section we are only concerned with the principal features of the technique sufficient to support a comparative assessment against the criteria presented in Table (3.1.1).

3.3.1 Phase Sequence Generation

The accumulation process can be considered as the DT integration of frequency to give phase. An M-bit accumulator generates a DT phase sequence whose frequency, f, is a linear function of a *phase-increment* input parameter, φ. For a sampling frequency, f_s, we have:

$$f = \frac{\varphi f_s}{2^M} \tag{3.3.1}$$

The frequency resolution is $\frac{f_s}{2^M}$ and defined by the sample frequency and accumulator word size alone. The phase sequence, $\phi(n)$, is given by:

$$\phi(n) = \langle \phi(n-1) + \varphi \rangle_{2^M} \tag{3.3.2}$$

We consider the development of Eqs. (3.3.1) and (3.3.2) in Chapter 4.

For a given sample frequency, arbitrarily fine frequency resolution is obtained by appropriate selection of accumulator word size, M. Following a change in φ, phase slope (and therefore frequency) transition occurs with a latency of one sample period and is precisely *phase-continuous*, which we consider further in Chapter 4. The phase-accumulator requires only adder and register elements and readily lends itself to pipelining and time-division multiplexing. The phase-increment can be generalised as a frequency control parameter, $F(n)$, that may be updated at the sample rate. Phase control is effected by adding a phase parameter, $\Phi(n)$, modulo 2^M to the phase accumulator output before phase-mapping. Using an M-bit adder effects the modulo 2^M operation. The technique is illustrated in Figure (3.3.1) using a length 2^M lookup table (LUT). The output sample is given by $y(n) = A(n)\sin\left(\frac{2\pi}{2^M}F(n)nT + \frac{2\pi}{2^M}\Phi(n)\right)$.

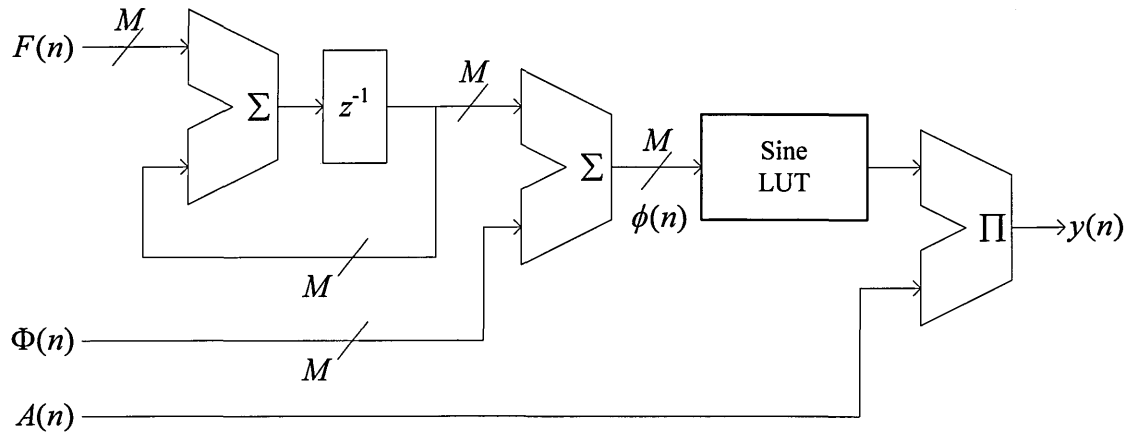


Figure (3.3.1): The phase-accumulating sinusoidal oscillator process model.

Phase-accumulators using unsigned integer arithmetic produce phase sequences with only positive slope and values bound on the interval $[0, 2^M - 1]$ that overflow and wrap around to zero. Alternatively, 2's complement integer arithmetic, as suggested by Moore [1977], produce phase sequences with positive or negative slope and values

bound on the interval $[-2^{M-1}, 2^{M-1}-1]$. Positive or negative phase-increments generate corresponding phase sequences with positive or negative slopes. Phase sequences with negative slope *underflow* to positive full scale, precisely satisfying the mathematical requirements of a *negative* frequency, which is essential in FM synthesis applications [Chamberlin, 1985; Chowning, 1973]. Neither arithmetic produces *radian-based* phase arguments as required by the sine function which is defined on the interval $[0, 2\pi)$. Multiplying $\phi(n)$ by $\frac{2\pi}{2^M}$ gives radian-based phase values, bound on the interval $[0, 2\pi)$ or $[-\pi, \pi)$ for unsigned or 2's complement arithmetic respectively. Real-time execution of this multiply operation is unnecessary with lookup table phase mapping since it is performed during the *pre-computed* sine function tabulation.

3.3.2 Sinusoidal Phase-mapping by Table Lookup

The simplest phase-amplitude mapping (so-called *phase mapping*) uses a lookup table containing a tabulated sine function [Tierney *et al*, 1971]. The lookup table contains one cycle of a unit-amplitude sinusoid, tabulated across L equally spaced phase points and may be defined as a vector, **S**, whose value, $\mathbf{S}[a]$, at address, a, is given by:

$$\mathbf{S}[a] = \sin\left(\frac{2\pi a}{L}\right), \quad a \in [0, L-1] \qquad (3.3.3)$$

Error-free phase mapping is realised when the lookup table contains 2^M tabulated samples, thereby mapping all accumulator phase states to a *unique* amplitude value, assuming sufficient resolution in amplitude quantisation. Under these conditions, the lookup table output samples are *precisely* equivalent to those of a generalised DT sinusoid for *all* phase-increment values.

Values of M required to ensure sufficient frequency resolution in computer music applications preclude this approach since lookup table lengths become excessive (M is

of order 24 bits). Truncating the M-bit phase value to the I most significant bits permits smaller lookup tables of length $L = 2^I$, but introduces amplitude errors since the accumulator phase states are no longer uniquely mapped into the amplitude domain. The residual $M - I$ bit-field represents the fractional distance between adjacent lookup table values. Amplitude errors associated with truncated phase mapping manifest in the frequency domain as components inharmonically related to the fundamental. The magnitude, frequency and number of these components are related to the degree of truncation *and* phase-increment [Nicholas and Samueli, 1987].

Moore [1977] considers phase-mapping errors in terms of an overall signal-to-noise ratio (SNR) which we consider further in Chapter 5 as a precursor to algorithm simulation and qualitative performance assessment. This SNR is based upon an error defined as the difference between a reference DT sinusoid computed to full-precision and the phase-accumulated table lookup approximation, computed over a large number of samples with various phase-increments.

The SNR of the synthesised signal is a function of lookup table length, 2^I, and the number of bits (excluding sign-bit), $b-1$, used to represent the tabulated samples. Moore [1977] suggests an architecture having $b-1=I$ gives a worst case SNR of approximately $6(I-2)$ dB, which cannot be improved further by making $b-1>I$ because of phase quantisation caused by discarding the $M-I$ bit-field. Alternatively, rounding the I-bit phase value to the nearest sample using the $M-I$ phase fraction bits improves SNR by approximately 6 dB, and requires $b-1=I+1$ giving a worst case SNR of $6(I-1)$ dB [Moore, 1977].

Truncation errors can be reduced by interpolating the lookup table access using the discarded $M-I$ phase bits as an implicit *fractional* address through appropriate scaling. The simplest interpolation technique is linear interpolation of the lookup table,

S. We define a linearly interpolated sample, $y(a, \alpha_a)$, at table address, a, and fractional address, α_a, as:

$$y(a, \alpha_a) = \mathrm{S}[a] + \alpha_a(\mathrm{S}[a+1] - \mathrm{S}[a]) \qquad (3.3.4)$$

where $a = 0, 1, 2, \ldots L-1$ and $\alpha_a \in [0, 1)$. Denoting the integer and fraction bit-fields of $\phi(n)$ as $\phi_I(n)$ and $\phi_F(n)$ respectively, we can express Eq. (3.3.4) in terms of $\phi(n)$ as:

$$y(n) = \mathrm{S}[\phi_I(n)] + \alpha(n)(\mathrm{S}[\phi_I(n)+1] - \mathrm{S}[\phi_I(n)]) \qquad (3.3.5)$$

where $\alpha(n) = \dfrac{\phi_F(n)}{2^{M-I}} \in [0, 1)$.

Moore [1977] suggests a linear interpolating architecture having $b-1 = 2(I-1)$ gives a worst case SNR of approximately $12(I-1)$ dB, halving I with respect to the non-interpolated case for the same SNR. As I increases, the memory savings become considerable using this method. It is evident from Eq. (3.3.4) that linear interpolation requires a single multiplication, three add/subtract and two table lookup operations. The technique is depicted in Figure (3.3.2).

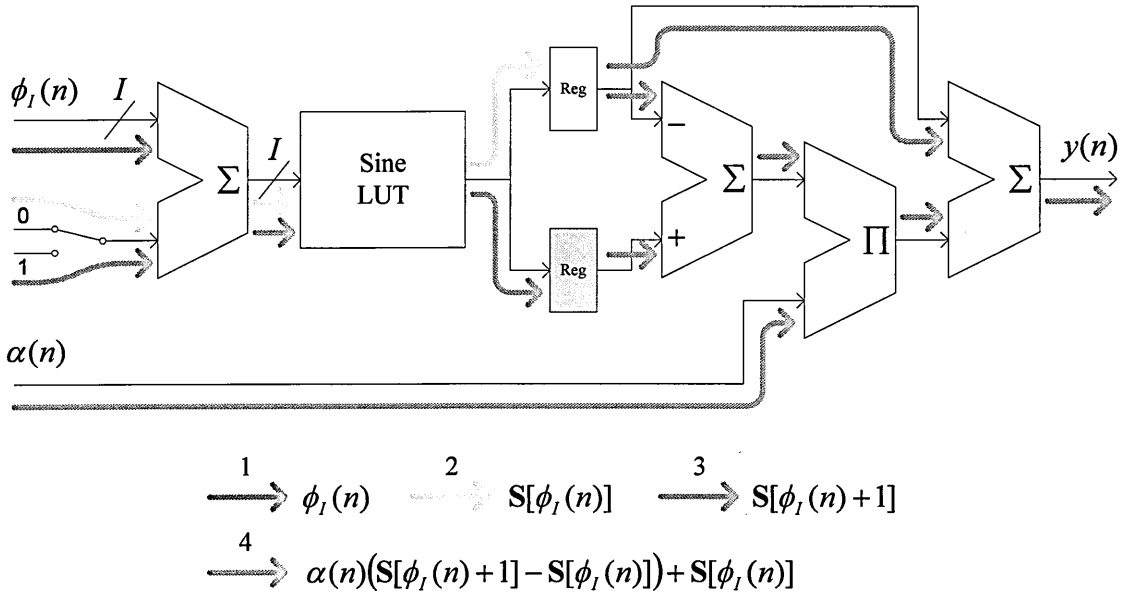


Figure (3.3.2) Linearly interpolated phase mapping using multiplexed access of a single lookup table.

There are principally two architectures for hardware implementation of linearly interpolated phase-mapping. A single lookup table can be accessed twice as depicted in Figure (3.3.2) and the first-order difference, $S[\phi_I(n)+1]-S[\phi_I(n)]$, computed. Alternatively, two lookup tables can be addressed in parallel, with the second containing pre-computed first-order difference samples as depicted in Figure (3.3.3) [Snell, 1977]. In all cases, an additional multiplication by $A(n)$ is required to effect amplitude control. Linear interpolation only approximates the amplitude value at a particular phase. We consider lookup table interpolation further in Chapter 4 and present a phase mapping technique that achieves ideal performance with $L << 2^M$.

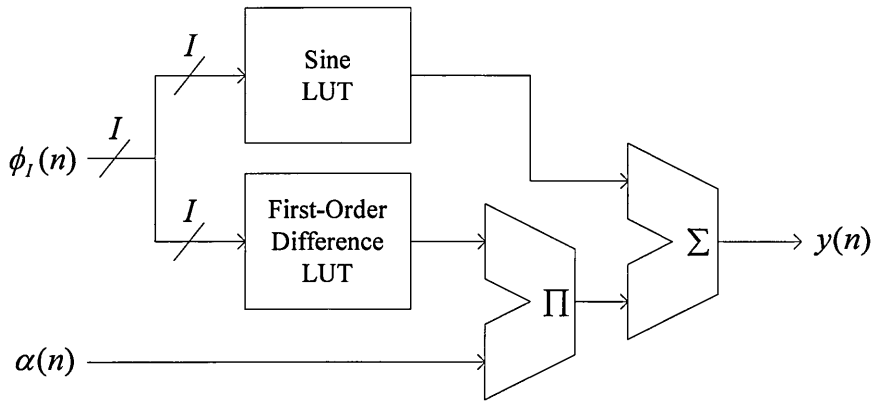


Figure (3.3.3) Linear interpolation using two lookup tables to eliminate consecutive access of a single memory.

3.3.3 Truncated Taylor Series Sinusoidal Phase-mapping

We can implement the phase-mapping operation directly by computing $\sin(\phi(n))$ using a Taylor series of order k, given by:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots (-1)^{k-1} \frac{x^{2k-1}}{(2k-1)!} \qquad (3.3.6)$$

$$x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

The series converges rapidly to an accuracy that is governed by the number of terms, k. For a 4th order series ($k=4$), Eq. (3.3.6) can be factorised using Horner's algorithm [Orfanidis, 1996] to give:

$$\sin(x) \approx \left(\left(\left(\left(\frac{-1}{7!}\right)x^2 + \frac{1}{5!}\right)x^2 - \frac{1}{3!}\right)x^2 + 1\right)x \qquad (3.3.7)$$

$$x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

Eq. (3.3.7) represents the 4th order Taylor series requiring the minimum number of arithmetic operations. In general, a k term factorised series requires $k+1$ multiplies, $k-1$ additions and $k-1$ constants. A further multiply is required to normalise the phase-accumulator argument to modulo-2π form. The Taylor series is only defined on the two quadrant interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, the other quadrants are generated using trigonometric identities, which adds computational overhead. We can assess the performance of this method by defining an error function as the difference between the k^{th}-order Taylor series and $\sin(x)$ computed to full-precision. Figure (3.3.4) shows the error variation measured in dB, against the phase argument, x, for several values of k. Maximum amplitude error occurs when $x = \pm\frac{\pi}{2}$ and requires $k \geq 5$ to be below –96dB, comparable with a 16-bit quantisation noise floor. A 5th-order Taylor series requires 6 multiply operations making it prohibitive compared to other techniques.

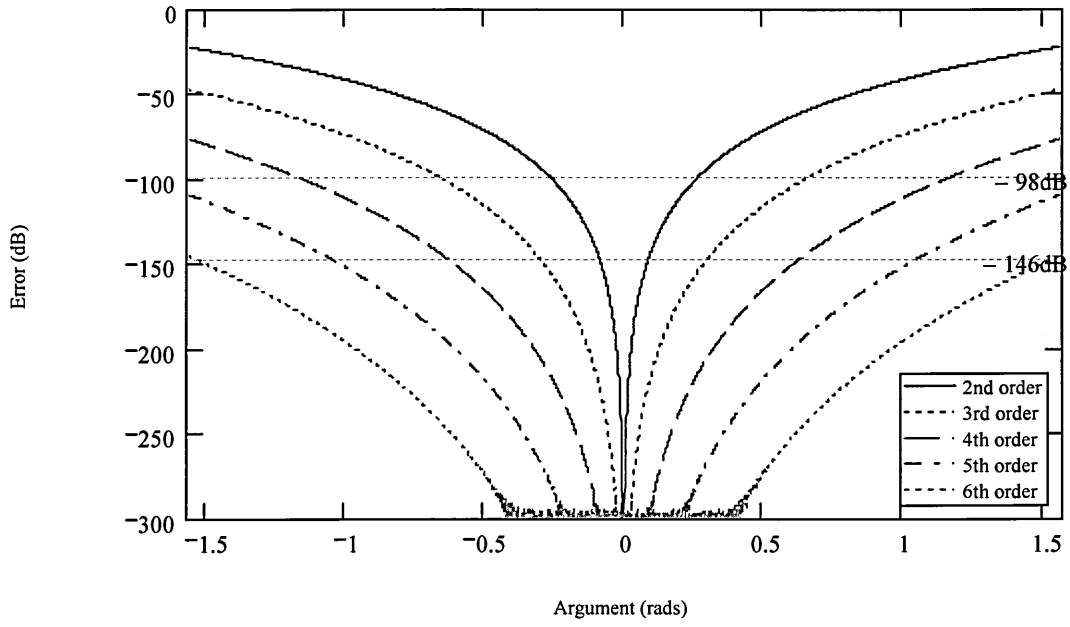


Figure (3.3.4): Error function for various order Taylor series approximations of $\sin(x)$. *(16 and 24-bit quantisation noise floor levels are shown for reference.)*

3.4 The CORDIC Algorithm

In this section we review utilisation of the CORDIC algorithm to solve the sinusoidal phase-mapping problem. Since it is a technique not specifically developed for frequency synthesis phase-mapping, we first develop its underlying principles using the concept of a vector rotation.

3.4.1 The CORDIC Algorithm as a Vector Rotation

The CORDIC (COordinate Rotation DIgital Computer) algorithm uses a convergent iteration process to compute the rotation of a vector in a Cartesian coordinate system. The technique was first presented by Volder [1959] in connection with efficient (bit-serial) airborne computation, and later unified by Walther [1971]. The CORDIC method is based on the simple observation that a unit length vector, $[1 \quad 0]^T$, rotated by θ, has an end point $[\cos(\theta) \quad \sin(\theta)]^T$. The CORDIC transformation is computed over m iterative steps, each involving a 'partial rotation' by some fraction of θ.

The rotation of a vector, $[x_0 \quad y_0]^T$, by an angle, θ, in Cartesian coordinates produces the vector $[x_m \quad y_m]^T$, and can be represented by the matrix operation:

$$\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{3.4.1}$$

Using the trigonometric identity $\cos(\theta) = \dfrac{1}{\sqrt{1+\tan^2(\theta)}}$, Eq. (3.4.1) can be expressed as:

$$\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2(\theta)}} \begin{bmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{3.4.2}$$

In the CORDIC method, rotation by θ is implemented by a number of partial rotations, α_i. Any angle, θ, within a defined interval can be represented to a particular accuracy by a weighted sum of m partial angles, α_i, with unitary weights, $\sigma_i \in \{-1, 1\}$, thus:

$$\theta = \sum_{i=0}^{m-1} \sigma_i \alpha_i \tag{3.4.3}$$

where $\sigma_i \in \{-1, 1\}$. The magnitude of α_i decreases with index, i. The initial α_i values are weighted positively until the sum exceeds θ, whereupon the α_i values are negatively weighted until the sum falls below θ. This process is repeated for m iterations until a specified accuracy is reached. The sign of the difference between θ and the sum given by Eq. (3.4.3) controls the value of σ_i.

The partial angles, α_i, are chosen according to:

$$\tan(\alpha_i) = 2^{-i} \qquad i = 0, 1, 2, 3, \ldots m-1 \tag{3.4.4}$$

An auxiliary variable, z_i, represents the accumulated partial angles and is used to control the value of σ_i. As m increases, z_i tends to 0. For $z_0 = \theta$ we have:

$$\begin{aligned} z_{i+1} &= z_i - \sigma_i \tan^{-1}(2^{-i}) \\ \sigma_i &= \begin{cases} +1 & z_i \geq 0 \\ -1 & z_i < 0 \end{cases} \end{aligned} \tag{3.4.5}$$

The CORDIC rotation is not a pure rotation but a *rotation-extension* since the magnitude of the rotated vector increases as the rotation proceeds. This necessitates introduction of a scaling factor to keep the vector at constant magnitude. Eq. (3.4.2) can now be written as the matrix difference equation:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = k_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{3.4.6}$$

where the scaling factor, k_i, is given by $k_i = \frac{1}{\sqrt{1+2^{-2i}}}$, and can be generalised for m iterations to a single value given by:

$$k = \prod_{i=0}^{m-1} \frac{1}{\sqrt{1+2^{-2i}}} \tag{3.4.7}$$

k approaches a limiting value of approximately 0.607253 as $m \to \infty$.

We can move the k_i scaling operations to the end of the iteration process, combining them into a single multiplication by k. The iteration difference equations without the k_i scaling term become:

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i 2^{-i} y_i \\ y_{i+1} &= y_i + \sigma_i 2^{-i} x_i \\ z_{i+1} &= z_i - \sigma_i \tan^{-1}(2^{-i}) \end{aligned} \tag{3.4.8}$$

where $i = 0, 1, 2, \ldots m$.

Eqs. (3.4.8) require add, subtract, and arithmetic bit shift operations only. A table-lookup operation provides each $\tan^{-1}(2^{-i})$ value. The principal advantage of the CORDIC algorithm arises from the simplicity of these operations, especially when considering VLSI implementation. After m iterations, the x_m and y_m values given by Eq. (3.4.8) must be multiplied by k to give the correct result, thereby involving a costly multiply operation. Alternatively, selecting the initial values $x_0 = k$, $y_0 = 0$ and $z_0 = \theta$, eliminates this multiply operation with $x_m = \cos(\theta)$ and $y_m = \sin(\theta)$ after m iterations.

This process is illustrated in Figure (3.4.1) computing $\sin\left(\frac{\pi}{8}\right)$ over 15 iterations.

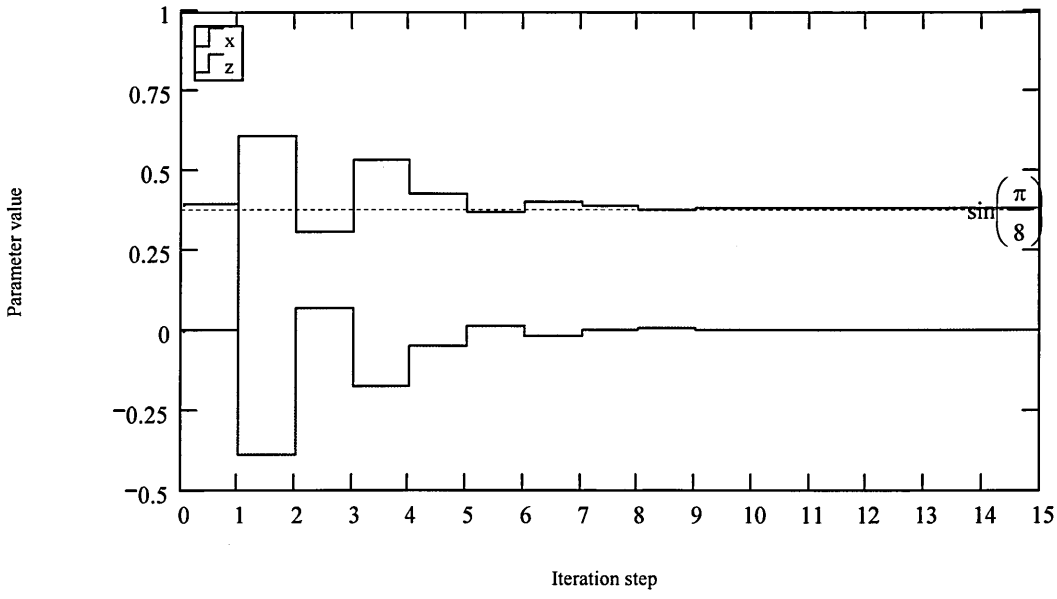


Figure (3.4.1): An example of the CORDIC algorithm computing $\sin\left(\frac{\pi}{8}\right)$ *over 15 iterations. The red trace indicates the convergence to* $\sin\left(\frac{\pi}{8}\right)$ *as the iteration proceeds.*

Since $\tan^{-1}(2^{-i}) \approx 2^{-i}$ for large i, b bits of phase-mapping precision can be obtained with b iterations. The process remains convergent on the interval $\theta \in [-r, r]$, where $r = \sum_{i=0}^{\infty} \tan^{-1}(2^{-i}) \approx 1.743286$ radians. For convenience, we take the region of convergence as $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Values of θ outside this interval can be processed by using the trigonometric identities: $\cos(x \pm 2n\pi) = \cos(x)$, $\sin(x \pm 2n\pi) = \sin(x)$, $\cos(x - \pi) = -\cos(x)$ and $\sin(x - \pi) = -\sin(x)$, where $n = 1, 2, 3, \ldots$, to bring θ within the convergent interval.

3.4.2 CORDIC Application in Digital Sinusoidal Oscillators

Rotation of the vector $[1 \quad 0j]^T$ by $\phi(n)$ is equivalent to the discrete-time, unit-amplitude, complex sinusoid $(\cos(\phi(n))+\sin(\phi(n))j)$. The rotation is therefore mapping from the phase to amplitude domain. The so-called *rotation form* CORDIC algorithm, outlined in section (3.4.1), performs this mapping and requires b iterations for b bits of phase resolution. It is *not* a synthesis 'from scratch' technique and requires prior computation of a phase sequence, $\phi(n)$, using phase accumulation as discussed in section (3.3).

Application of the CORDIC algorithm requires partitioning of an M-bit phase word, $\phi(n)$, into two fields. Denoting the most significant bit (MSB) of $\phi(n)$ as $\phi(n)_{M-1}$, the two bit field, $(\phi(n)_{M-1}, \phi(n)_{M-2})$, represents the phase quadrant containing $\phi(n)$. The B-bit field, $(\phi(n)_{M-2}, \phi(n)_{M-3}, \ldots \phi(n)_{M-B-1})$, represents the z_0 phase value which initialises the CORDIC algorithm. Maximum phase resolution is obtained when $B = M-1$, otherwise the phase word is truncated when $B < M-1$. Figure (3.4.2) shows the architecture of a CORDIC phase mapping processor including the $\phi(n)$ bit-field partitioning. The complement blocks are controlled by the quadrant bit-field $(\phi(n)_{M-1}, \phi(n)_{M-2})$ to correctly reconstitute the sinusoid.

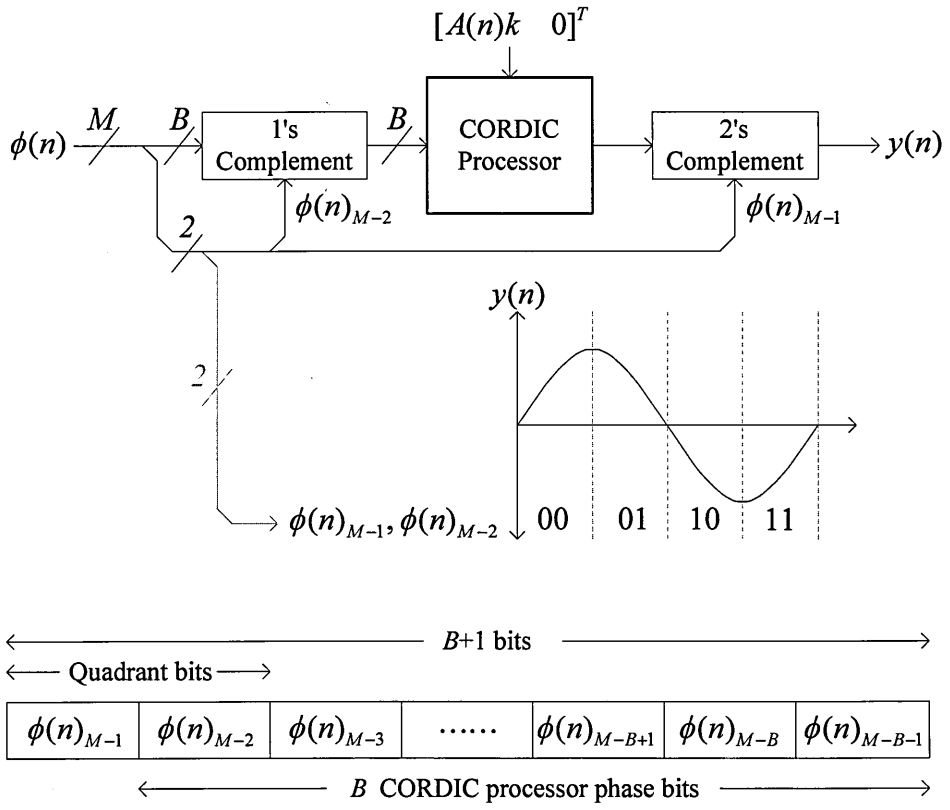


Figure (3.4.2) CORDIC phase mapping architecture.

The phase accumulator output is bound according to $\phi(n) \in [0, 2^M - 1]$. Since the CORDIC algorithm expects z_0 in radians we modify Eq. (3.4.5) thus:

$$z_{i+1} = z_i - \sigma_i \left(\frac{2^B}{\pi} \right) \tan^{-1}(2^{-i}) \tag{3.4.9}$$

$$\sigma_i = \begin{cases} +1 & z_i \geq 0 \\ -1 & z_i < 0 \end{cases}$$

The algorithm is initialised with $[x_0 \quad y_0]^T = [kA(n) \quad 0]^T$ to obtain a sinusoid with amplitude $A(n)$.

The CORDIC algorithm typically uses fixed-point arithmetic within each processing stage. Barrel-shift operations with a fixed word size introduce truncation errors at each stage and leads to degraded SNR for small sinusoid amplitudes. A CORDIC

architecture using floating-point arithmetic has been suggested [Phillips, 1997] which mitigates truncation errors at the expense of increased computational complexity.

3.4.3 Sequential and Recursive CORDIC Implementation

The CORDIC algorithm can be implemented in essentially two ways – a *sequential* pipeline of m processing elements, each performing a sub-rotation in accordance with Eq. (3.4.8) or a *recursive* utilisation of a single processing element iterated m times. The recursive approach is problematic in multiplexed systems computing one sinusoid sample per clock cycle since process clocking is increased m-fold. The pipelined architecture incurs an additional inter-stage register overhead but does not require an explicit barrel-shifter function within each stage. This operation is effected by a hardwired interconnect between the process stages. However, sinusoid computation now has a latency of m sample clock cycles and is problematic for large m as required for precise phase-mapping. Figure (3.4.3) illustrates both forms, with the processing element detail shown in Figure (3.4.4). One table lookup, three addition/subtraction, and two barrel shift operations are performed in parallel within each stage.

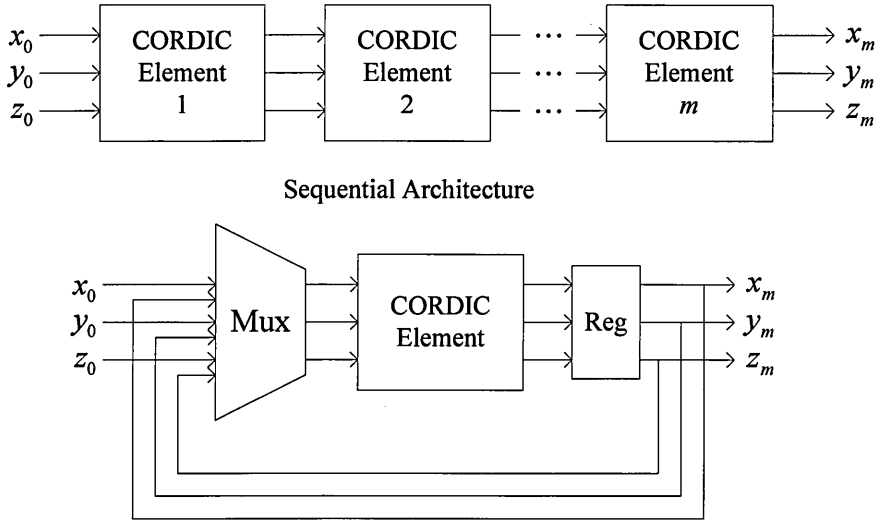


Figure (3.4.3) Sequential and recursive implementation of the CORDIC algorithm.

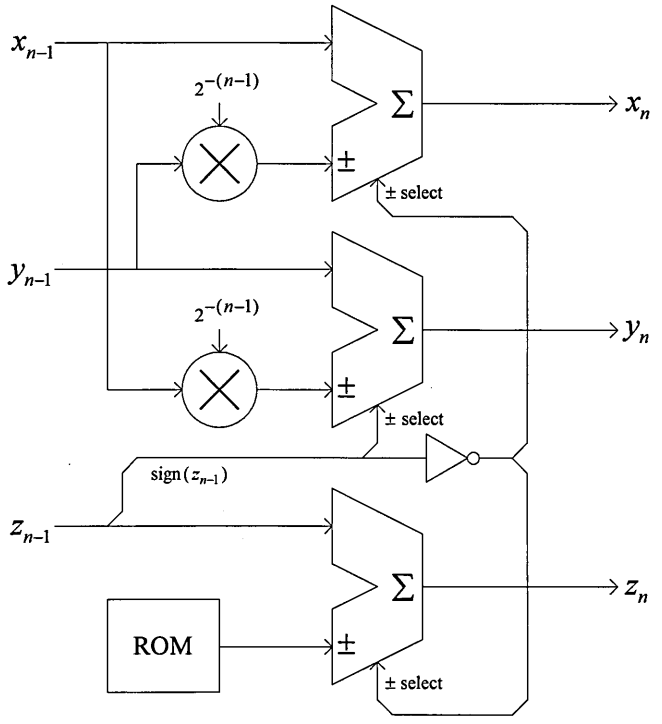


Figure (3.4.4): CORDIC processing element architecture. This represents the n^{th} of m stages for a sequential architecture and a single stage within a recursive architecture.

A floating-point CORDIC algorithm [Phillips, 1997] represents an alternative to sinusoidal phase-mapping using interpolated table lookup. Furthermore, the arithmetic simplicity of component operations suggest the technique is attractive to VLSI implementation. Madisetti *et al* [1999] describe a phase-accumulating oscillator using CORDIC phase-mapping which is based upon the sequential architecture outlined above. However, VLSI design and control complexities favour the interpolated table lookup approach since it comprises highly optimised standard components [Phillips, 1997] and incurs less computational latency. In Chapter 5 we present a phase mapping technique which achieves ideal performance (SNR bound by quantisation noise) with greatly reduced lookup table length.

3.5 Conclusions

The number of arithmetic operations, state variables and control parameters are important metrics which decide complexity and throughput performance in multiple oscillator embodiments. Assessment metrics P1 to P6 are summarised in Table (3.1.1). Multiplication is the principal arithmetic operation due to its relatively high execution time and larger VLSI area requirement. All recursive forms require two state variables and between one and four control coefficients in contrast to the single state variable and phase increment parameter of the phase-accumulating oscillator. All recursive oscillators exhibit a non-linear relationship between oscillation frequency and the control parameter(s) (property P5 of Table (3.1.1)). Linear frequency control imposes additional sine/cosine operations with a scaled argument, greatly increasing computational overhead.

The direct-form oscillator requires only one multiplication and one addition operation (property P1) and suffers no exponential amplitude instability. A linear increase in the round-off error variance over time is typical necessitating periodic re-initialisation to maintain SNR below a predefined level (property P3). Phase-continuous frequency transition requires a new IC computation comparable in complexity to the waveguide-form amplitude normalising multiplication (properties P4 and P6). The coupled-form requires four multiplication and two addition operations (property P1) and has inherently unstable oscillation amplitude (property P3), requiring periodic re-initialisation to maintain amplitude drift within a predefined error bound. Frequency transition is phase-continuous with no re-computation of IC values necessary (property P6). The coupled-form exhibits a uniform pole distribution around the unit-circle causing a corresponding uniform distribution of frequencies across the Nyquist interval. The modified coupled-form requires two multiplication and two addition operations but

provides phase-continuous frequency transition (property P6) and unconditionally stable amplitude (property P3). SNR is bound only by quantisation noise and low frequency control resolution is optimal. In contrast, the direct-form and waveguide-form both exhibit diminishing control resolution at low frequencies for a particular coefficient quantisation interval. The waveguide-form requires two multiplication and three addition operations and provides phase-continuous frequency transition with unconditionally stable amplitude (property P3). However, direct computation of the amplitude normalising coefficient (property P4) is problematic. Table (3.5.1) summarises the performance attributes of the four recursive oscillators.

Property	Direct-Form	Coupled-Form	Modified Coupled-Form	Waveguide-Form
Multiplication overhead including $A(n)$ (P1)	2 *Note 1*	5(6) *Note 2*	3(4) *Note 2*	3 *Note 3*
Addition overhead (P1)	1	2	2	3
State variables (P2)	2	2	2	2
Frequency coefficient(s)	1	4	2	1
ICs	2	2	2	2
Phase-continuous frequency transition (P6)	No	Yes	Yes	Yes
Frequency transition ICs	1 *Note 4*	0	0	0
Normalisation multiplication	0	0	0	1 *Note 3*
Amplitude stability over n (P3)	Quasi-stable *Note 5*	Unstable	Stable	Stable *Note 3*
$F(n)$ – amplitude interaction (P4)	Yes *Note 6*	No	No	Yes *Note 3*
$F(n)$ control (P5)	Non-linear	Non-linear	Non-linear	Non-linear
Low-frequency control resolution	Low	Uniform over Nyquist	High	Low
$\Phi(n)$ control (P5)	Re-initialise	Re-initialise	Re-initialise	Re-initialise

1 We discount the multiplication by -1 as trivial.
2 A further multiply is only required for quadrature amplitude scaling.
3 An additional amplitude-normalising multiplication is required at every frequency transition.
4 Two ICs are required, but only one needs to be computed explicitly for constant amplitude.
5 Round-off error noise power increases as N_s, but there is no exponential amplitude drift.
6 Non-interactive frequency-amplitude control requires re-initialising with new ICs.

Table (3.5.1): Summary of recursive oscillator performance against metrics defined in Table (3.1.1).

The phase-accumulating oscillator requires a single addition and table lookup operation and has an intrinsically linear frequency control characteristic with resolution bound only by the accumulator word size. The technique provides access to the underlying phase-time function and forms the basis of advanced synthesis techniques, in addition to providing a means of precise phase control. The oscillator provides unconditionally stable amplitude irrespective of frequency and time. Phase-mapping errors due to truncated lookup table addressing manifest themselves as a degradation in SNR and can be reduced by interpolated table lookup. We have considered direct computation of the sine function and found truncated Taylor series computations excessive in terms of the number of multiplication operations required for reasonable phase mapping precision. CORDIC phase mapping significantly reduces the lookup table overhead at the expense of computational latency and is only applicable to *sinusoidal* phase-mapping.

We conclude that the phase-accumulating oscillator is the most flexible architecture from a synthesis and efficiency of implementation perspective. However, the direct-form and modified coupled-form recursive oscillators represent an interesting alternative in particular applications. In Chapters 4 and 5 we develop *wavetable lookup synthesis* based upon the phase-accumulating oscillator as a generic table lookup synthesis technique using sinusoidal and non-sinusoidal phase mapping functions. The inherent arithmetic partitioning of the phase-accumulating oscillator enables processing in the *phase domain* and underpins a subclass of computationally efficient harmonic and partial based additive synthesis techniques which we investigate in Chapter 6. We define phase domain processing as the algorithmic modification of phase information, prior to the phase-amplitude mapping process, to effect partial frequency or phase control.

Chapter 4 Wavetable Lookup Synthesis

4.1 Background

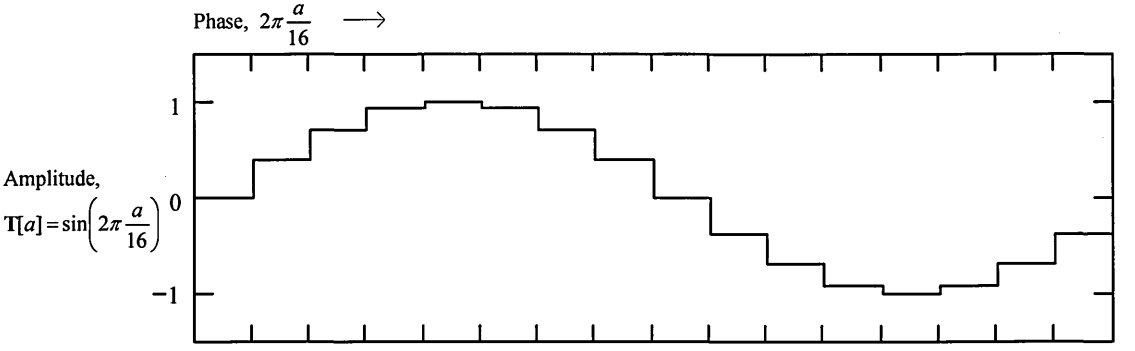
In Chapter 3 we introduced the phase accumulation oscillator as a basis for synthesising both sinusoidal and non-sinusoidal signals using a phase-amplitude mapping function based on table lookup. This technique provides greater flexibility compared with recursive algorithms which are restricted to the synthesis of sinusoidal signals and hampered by nonlinear control characteristics and the need for periodic re-initialisation in some cases. Phase accumulation synthesis is fundamentally a two-stage process – phase sequence generation and phase-amplitude mapping. This approach underpins a synthesis paradigm built on discrete phase-amplitude mapping that we generalise as *wavetable lookup synthesis* (WLS). A subclass of WLS involves direct manipulation of the phase sequence to effect frequency scaling or phase shifting (phase domain processing) prior to phase-amplitude mapping, which we investigate in Chapter 6. Chapter 3 introduced phase accumulation and *sinusoidal* phase-mapping algorithms. However, WLS promises the synthesis of signals with time-varying, multiple harmonic spectra.

Chapter 4 reviews and develops WLS as applied to the generation of musical signals. WLS begins with a DT phase sequence, whose slope represents frequency, which is then mapped to the amplitude domain using a lookup table or *wavetable*. The fundamental premise for WLS is that table lookup operations are *much* faster than algorithmically computing the tabulated samples from scratch in real time. The computational efficiency of WLS therefore increases with the number of arithmetic operations that would be needed to compute each tabulated value in real time.

We begin by presenting a taxonomy of wavetable classes currently prevalent in the literature [Roads, 1996] and review the concept of resampling tabulated signals to effect frequency control. We investigate frequency as the time rate-of-change of phase which leads naturally to phase accumulating frequency synthesis and the concept of phase-continuity. We conclude by investigating optimal phase mapping and the special case when a wavetable contains a complete sampled sound - so-called *sampling synthesis* as introduced in Chapter 2.

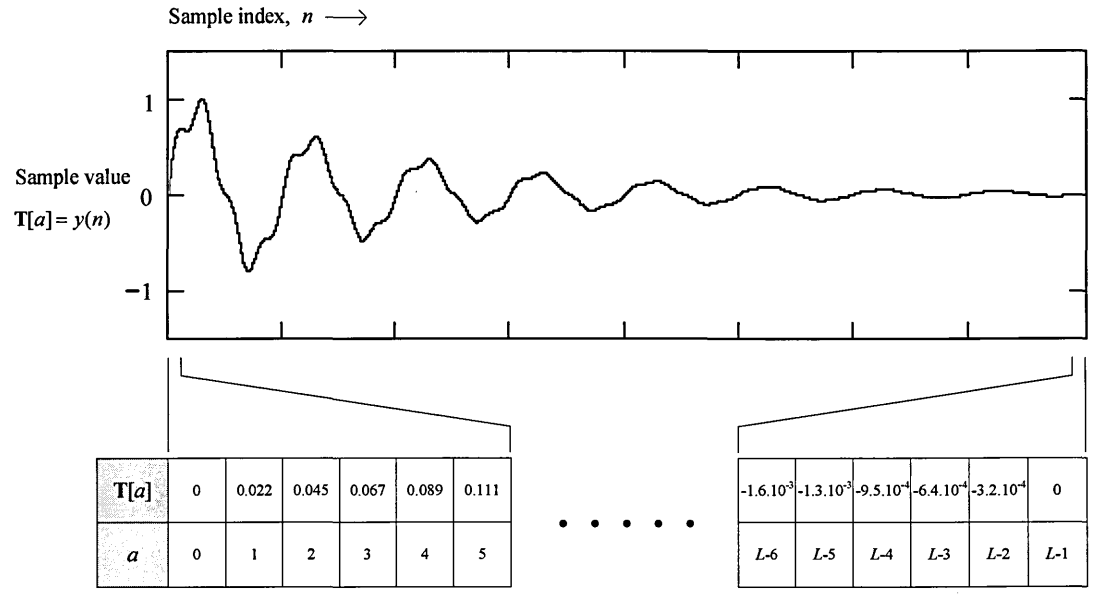
4.1.1 Foundations of Wavetable Lookup Synthesis

A wavetable is a list of regularly time-sampled, amplitude-quantised signal values stored in consecutive memory locations as a sequence of numbers. The signal can be a single cycle of a periodic function (e.g. a sinusoid) or many cycles of a complex quasi-periodic signal extending over an arbitrary period of time. Fundamentally, a wavetable performs a discrete-time (DT) translation from the phase to amplitude domain – the so-called phase mapping function. The tabulated sample values are necessarily quantised in amplitude requiring a particular number of bits in the memory word and number representation format. (e.g. 16 bit, fixed-point 2's complement.) The samples may be computed in non real-time from harmonic amplitude and phase data using a Fourier series summation [Chamberlin, 1985] or obtained directly by sampling a sound signal and quantising using an analogue-to-digital converter – so called 'sampling synthesis' [Roads, 1996]. We define two distinct wavetable classes which are exemplified in Figures (4.1.1a) and (4.1.1b) – *single cycle* wavetables containing precisely one cycle of a periodic function, where the beginning and end of the wavetable are phase-continuous, and *multi-cycle* wavetables containing an entire sampled sound or pre-computed waveform sequence.



T[a]	0	0.38	0.71	0.92	1	0.92	0.71	0.38	0	-0.38	-0.71	-0.92	-1	-0.92	-0.71	-0.38
a	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure (4.1.1a): A single-cycle wavetable tabulating precisely one cycle of a sinusoid over 16 samples. A coarse amplitude quantisation interval has been chosen for clarity. **T**[*a*] *represents the tabulated wavetable sample at address a.*



T[a]	0	0.022	0.045	0.067	0.089	0.111	• • • • •	$-1.6.10^{-3}$	$-1.3.10^{-3}$	$-9.5.10^{-4}$	$-6.4.10^{-4}$	$-3.2.10^{-4}$	0
a	0	1	2	3	4	5		L-6	L-5	L-4	L-3	L-2	L-1

Figure (4.1.1b): A multi-cycle wavetable tabulating a complex signal, $y(n)$, (e.g. a sampled sound) over L samples. **T**[*a*] *represents the tabulated wavetable sample at address a.*

All wavetables can be grouped into one of these two subclasses. Further classification is possible according to signal characteristics and the method used to compute the tabulated values as depicted in the taxonomy of Figure (4.1.2).

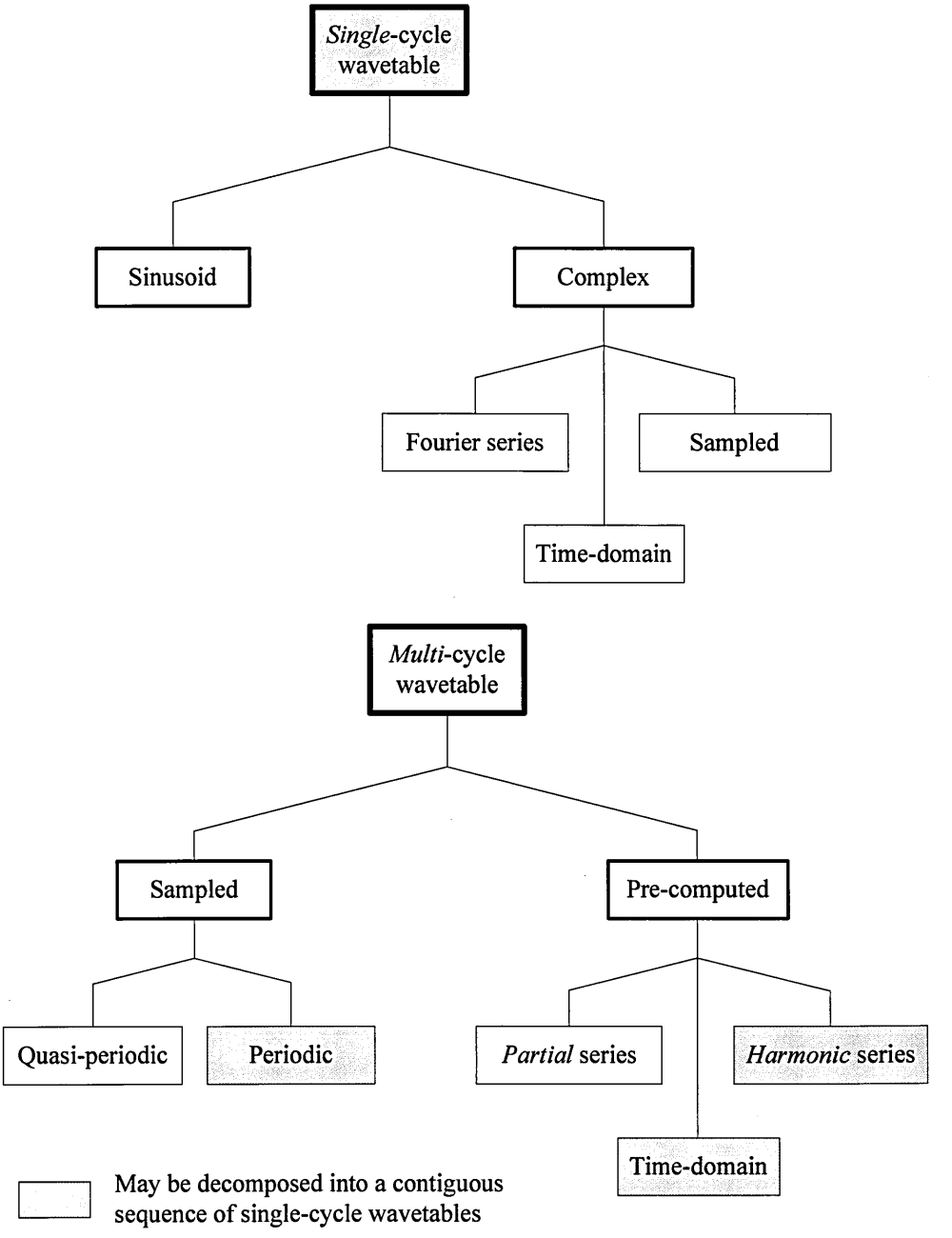


Figure (4.1.2): Single and multi-cycle wavetable classification.

The systematic *addressing* of a wavetable to resynthesise the tabulated signal at a particular frequency is the basis of all WLS. Single cycle wavetables are circular data

structures with beginning and end points precisely *phase-continuous* to ensure a discontinuity free transition when the table index wraps-around. They contain precisely one cycle of a periodic function and are addressed cyclically, modulo the table length, to generate a continuous signal at a particular frequency. Violation of the phase-continuity condition causes a corresponding amplitude discontinuity as the modulo addressing wraps around the wavetable end point.

Multi-cycle wavetables typically contain a sampled sound (e.g. an entire musical instrument note) although they can be pre-computed directly. Multi-cycle wavetables can be decomposed into a contiguous sequence of *single-cycle* wavetables, as illustrated in Figure (4.1.3), if the number of samples in the fundamental period of the tabulated signal equals the *single-cycle* wavetable length. This condition can only be satisfied if the fundamental and partial frequencies are constant over time and follow an harmonic distribution. This decomposition enables the resynthesised pitch and timbral evolution over time to be independently controlled.

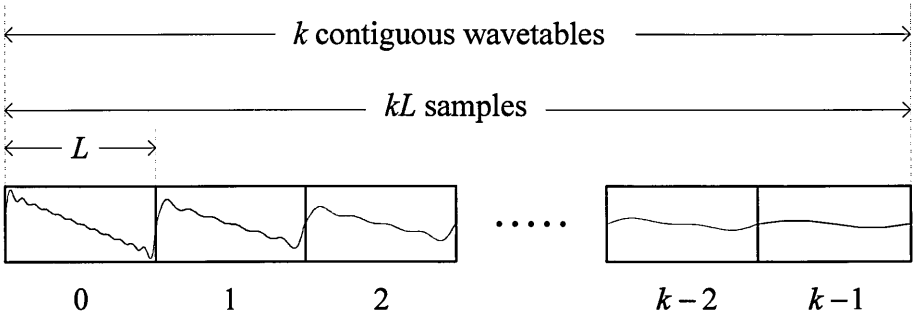


Figure (4.1.3): Decomposing a multi-cycle wavetable into a sequence of contiguous single-cycle wavetables which are accessed sequentially to synthesise time-varying timbre.

The physical wavetable memory space (comprising kL samples in the example of Figure (4.1.3)) is organised into a contiguous sequence of k wavetables. The wavetable address now comprises a *sample* component (the least significant bits) and a *table*

component (the most significant bits) which specifies the particular wavetable (and therefore timbre) being indexed by the phase accumulator. Quasi-periodic signals *cannot* be subdivided into contiguous single-cycle wavetables of fixed length without introducing discontinuities at the wavetable boundaries.

The signal-to-noise ratio (SNR) of the tabulated signal represents an important quality metric. We define SNR as the ratio of signal power to noise power, computed over the length of the wavetable, with noise including both stochastic and deterministic (i.e. unwanted) components. The SNR of a pre-computed wavetable signal is bound entirely by the signal-to-quantisation-noise ratio (SQNR) corresponding to the number of bits used to represent the sample values, the tabulated signal amplitude relative to full scale, and the probability density function (PDF) of the tabulated signal [Zőlzer, 1997]. We consider the SQNR of tabulated signals further in section (4.1.2). Multi-cycle wavetables obtained by sampling a sound invariably contain additional noise components due to incomplete suppression of spectral components above the Nyquist frequency prior to sampling. These components, whose magnitude can only be minimised, will alias into the Nyquist interval and can therefore be considered as unwanted noise-like components additional to amplitude quantisation noise. Pre-computed wavetables (both single cycle and multi-cycle) have SNR bound entirely by quantisation noise since their frequency domain characteristics can be band-limited to the Nyquist interval prior to tabulation. An important exception are wavetables specified in the time-domain (i.e. specified completely by *shape*). This easily introduces frequencies above the Nyquist frequency which will alias into the Nyquist region upon resynthesis and degrade SNR.

4.1.2 Wavetable Signal Tabulation

A wavetable is fundamentally a list of numbers which represent time-sampled signal amplitudes. In principle, this list can be generated in many ways. For example, a direct time domain specification (e.g. 'drawing' the desired waveform) or specification in the frequency domain as a weighted series of harmonically related sinusoids. Time domain specification allows the *shape* of the waveform to be precisely controlled and was one of the input techniques available in the Fairlight CMI series of computer music synthesis systems [Roads, 1996]. However, this method risks the introduction of discontinuities into the waveform, especially at the waveform end points, leading to frequency components above the Nyquist limit that will alias into the Nyquist interval upon resynthesis. However, the principal disadvantage of time domain specification (i.e. drawing or *shape* specification) is the limited correlation between waveform shape and perceived timbre which bears a much stronger relation to the harmonic structure of the signal [Chamberlin, 1985, Moore, 1990]. Sampling real musical signals (e.g. a complete musical instrument note) is an alternative time domain wavetable filling technique, generally known as 'sampling synthesis' in the literature [Roads, 1996]. Assuming the sampled signal is appropriately bandlimited prior to sampling and quantisation, there can be no tabulated frequency components above the Nyquist limit, unlike direct time-domain specification. We consider sampling synthesis further in section (4.2.6).

We now consider the tabulation of a precisely periodic signal using a concise analytical definition where there are two dimensions to consider – phase and amplitude. In the digital domain both of these dimensions necessarily take on discrete values. When computing samples of a particular signal in real-time, the independent phase variable is a function of discrete time, nT, where n is the sample index and T is the sampling

period. However, the non real-time nature of signal tabulation causes the phase variable to become a function of the wavetable address variable. Similarly, the nature of digital storage forces the dependent amplitude variable to become discrete, or quantised according to the number of bits available in the wavetable memory locations. We begin by considering the phase of a tabulated signal.

A signal, $f(x)$, is periodic with period τ if $f(x) \equiv f(x+\tau)$, where x denotes the phase variable. For the simple case when we tabulate the sine function, $f(x) = \sin(x)$, we note the periodicity condition is satisfied when $\tau = 2\pi$. The sine function should therefore be tabulated on the interval $x \in [0, 2\pi)$ which is mapped to the wavetable address range $[0, L-1]$, where L is the wavetable length. To effect this mapping for a particular wavetable address, a, we must preserve the equality $\frac{x}{2\pi} = \frac{a}{L}$ when computing the tabulated values of $f(x)$ to ensure phase continuity at the wavetable boundaries. The phase argument of our periodic signal, $f(x)$, is therefore given by $x = 2\pi\frac{a}{L}$, $a \in [0, L-1]$, where a is analogous to the phase of the tabulated signal with a resolution of $\frac{2\pi}{L}$ radians. Sine and cosine wavetables of length L are defined thus:

$$\mathbf{T}[a] = \sin\left(\frac{2\pi}{L}a\right), \quad a \in [0, L-1], \quad L \in \mathrm{Z} \tag{4.1.1}$$

$$\mathbf{T}[a] = \cos\left(\frac{2\pi}{L}a\right), \quad a \in [0, L-1], \quad L \in \mathrm{Z} \tag{4.1.2}$$

where $\mathbf{T}[a]$ denotes the a^{th} location of the vector $\mathbf{T}$ which represents the wavetable. Wavetables generated using Eqs. (4.1.1) and (4.1.2) satisfy $\mathbf{T}[a] \equiv \mathbf{T}[a+L]$ and $\mathbf{T}[L] \equiv \mathbf{T}[0]$. The end and beginning samples of the wavetable are therefore phase-continuous as required.

It is informative to compare a single cycle sine wavetable defined according to Eq. (4.1.1) with a generalised DT sinusoid given by $s(n) = \sin\left(2\pi n \frac{f}{f_s}\right)$, with frequency, f, sample rate, f_s, and time index, n. The ratio $\frac{f_s}{f}$ is equivalent to the wavetable length, L, assuming a single cycle is tabulated and is the *only* frequency related parameter preserved in the tabulation of a periodic signal – the absolute values of sample rate and sinusoid frequency are lost. The time index, n, is equivalent to the wavetable address, a, over the interval $[0, L-1]$.

We extend the tabulation of sinusoids to define a single cycle, multi-harmonic wavetable vector using the inverse discrete Fourier transform (IDFT) to effect harmonic additive synthesis. The IDFT may be recast in non-complex form allowing the wavetable values to be specified as a function of wavetable address, a, the highest harmonic number, N_h, and the harmonic amplitude and phase vectors, A and Φ, respectively. We have:

$$\mathbf{T}[a] = \sum_{k=0}^{N_h} A_k \cos\left(2\pi \frac{a}{L} k + \Phi_k\right)$$

$$a \in [0, L-1] \quad k \in [0, N_h] \qquad (4.1.3)$$

$$N_h \in [1, \frac{L}{2} - 1]$$

where $A_k \in [0,1]$ and $\Phi_k \in [0, 2\pi)$ are the k^{th} harmonic amplitude and phase coefficients respectively and L is the wavetable length in samples. By choosing appropriate amplitude and phase values for each harmonic component, any single cycle periodic function can be tabulated, subject to the bound imposed by the Nyquist limit on the highest harmonic number in the series. The number of samples per cycle in the

fundamental is simply L. Therefore, a particular harmonic number, h, will have $\frac{L}{h}$ samples per cycle and so the highest harmonic multiplier in the series, N_h, will be bound by $N_h < \frac{L}{2}$ to satisfy the Nyquist sampling criterion.

Tabulating a signal (e.g. a sinusoid), necessarily involves amplitude quantisation according to the wavetable word size, b. We define the signal-to-quantisation-noise ratio, SQNR as:

$$\text{SQNR} = 10\log\left(\frac{\sigma_x^2}{\sigma_e^2}\right) \tag{4.1.4}$$

where σ_x^2 and σ_e^2 represent the signal and quantisation error variance, respectively. For a typical quantiser, defined by the function $Q(x)$, with quantisation interval q, input range $\pm x_{\max}$ and word-length b bits, we have $\sigma_e^2 = \frac{q^2}{12}$ and $q = \frac{2x_{\max}}{2^b}$. We define $Q(x)$, thus:

$$Q(x) = q\left\lfloor \frac{x}{q} \right\rfloor \tag{4.1.5}$$

with $Q(x) = 2^{1-b}\left\lfloor \frac{x}{2^{1-b}} \right\rfloor$ for $x_{\max} = 1$.

We now define a peak factor (ratio of peak to RMS value) $P_F = \frac{x_{\max}}{\sigma_x} = \frac{2^{b-1}q}{\sigma_x}$ and express the signal and quantisation error variances as $\sigma_x^2 = \frac{x_{\max}^2}{P_F^2}$ and $\sigma_e^2 = \frac{q^2}{12} = \frac{1}{3}x_{\max}^2 2^{-2b}$ [Zölzer, 1997]. We can now express Eq. (4.1.4) as:

$$\text{SQNR} = 10\log\left(\frac{3\left(2^{2b}\right)}{P_F^2}\right) = 6.02b - 10\log\left(\frac{P_F^2}{3}\right) \quad \text{dB} \tag{4.1.6}$$

A sinusoidal signal with $P_F = \sqrt{2}$ gives SQNR = $6.02b + 1.76$ dB. A signal with uniform probability density function (PDF) and $P_F = \sqrt{3}$ gives SQNR = $6.02b$ dB. Although not pertinent to a discussion on tabulating simple periodic signals, it is evident that *uncompressed* musical signals typically exhibit peak factors between 16 and 20 dB [Eargle and Foreman, 2002]. Therefore, sampling and quantising such signals for wavetable tabulation causes SQNR to be between 13 and 17 dB worse compared to the baseline value for sinusoidal signals. Indeed, all signals with $P_F > \sqrt{2}$ give a reduced SQNR upon quantisation compared to the sinusoidal baseline value.

4.1.3 Sampling a Tabulated Function

Consider a radix-2 length wavetable filled with L samples of a continuous-time (CT) sinusoidal signal, $x(t) = \cos(2\pi f_a t)$, of frequency f_a, uniformly sampled at f_{sa} samples per second. The wavetable therefore contains L samples of the corresponding DT sequence $x(n) = \cos(2\pi f_a n T_a)$, with each unit address increment corresponding to a sample interval of $T_a = \frac{1}{f_{sa}}$ seconds. The number of $x(n)$ cycles, N_c, contained in the wavetable is given by:

$$N_c = \frac{f_a L}{f_{sa}} \tag{4.1.7}$$

where $N_c \in [1, \frac{L}{2})$ to satisfy the Nyquist sampling theorem [Orfanidis, 1996] which requires more than two samples per cycle. If L is radix-2, then N_c always takes integer values preventing a discontinuity between the beginning and end of the wavetable. We therefore assume for this discussion that N_c takes on radix-2 integer values.

We resynthesise the tabulated signal at a new frequency, f, by cyclically addressing the wavetable with a $\log_2(L)$-bit counter whose value is incremented by φ at the read sample rate, f_{sb}. Wavetable output samples feed a digital to analogue converter (DAC) and reconstruction filter to generate the CT signal. Since the wavetable address, $a(n) = \phi(n) = \langle n\varphi \rangle_L$, is equivalent to the phase of the tabulated signal the wavetable provides a phase-amplitude mapping. This hypothetical arrangement is illustrated in Figure (4.1.4) where the wavetable is filled by setting $\varphi = 1$ and using f_{sa} to clock the address generator until L samples have been written.

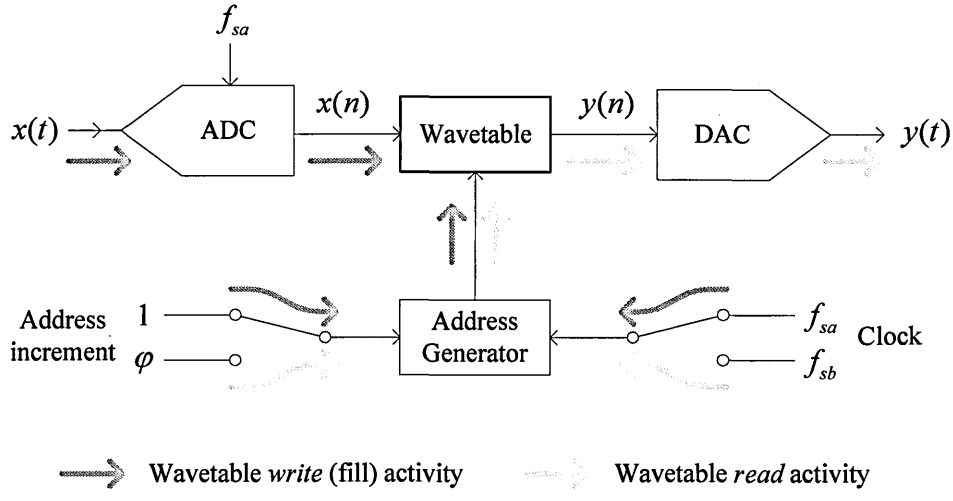


Figure (4.1.4): Hypothetical arrangement for tabulating a CT sinusoidal signal, $x(t)$, and resampling to effect resynthesis at a new frequency, $y(t)$.

During resynthesis, the wavetable is cycled every $\frac{L}{\varphi f_{sb}}$ seconds with output frequency, f, given by:

$$f = \frac{N_c \varphi f_{sb}}{L} \tag{4.1.8}$$

For $\varphi > 1$ the tabulated sinusoid is *effectively* sampled at a *lower* frequency than the original sampling frequency, f_{sa}. A unit address increment represents a time interval of

$T_a = \frac{1}{f_{sa}}$ seconds and so an increment of φ represents a sample interval of φT_a seconds or a new sampling frequency of $\frac{f_{sa}}{\varphi}$. For N_c tabulated cycles, we have $\frac{L}{N_c}$ samples per cycle and so $\varphi \in [1, \frac{L}{2N_c})$ to satisfy the Nyquist sampling theorem. We observe that the *samples per cycle* parameter, $\frac{f_{sa}}{f_a}$, is the *only* frequency related information preserved when a function is tabulated.

Combining Eqs. (4.1.7) and (4.1.8) to eliminate N_c we obtain:

$$f = \frac{\varphi f_a f_{sb}}{f_{sa}} \tag{4.1.9}$$

where $\varphi \in [1, \frac{L}{2N_c})$.

Assuming a unified sample rate we have $f_{sa} = f_{sb} = f_s$ so Eq. (4.1.9) becomes $f = \varphi f_a$ and substituting from Eq. (4.1.7) with $N_c = 1$, we obtain:

$$f = \varphi f_a = \frac{\varphi f_s}{L} \tag{4.1.10}$$

This discussion has described a rudimentary system for a digital oscillator which samples a tabulated signal stored in a wavetable with oscillation frequency constrained to *integer multiples* of the original signal frequency, f_a.

Let us now assume we have a single cycle of a sinusoid tabulated and so $N_c = 1$. It is clear that for a fixed sampling rate and integer address increments, we can only increase the output frequency relative to the minimum frequency, $\frac{f_s}{L}$, when $\varphi = 1$. For $\varphi > 1$, the wavetable is *decimated* and the tabulated sinusoid resampled at a *lower* sample frequency, f_s', given by:

$$f_s' = \frac{f_s}{\varphi} \tag{4.1.11}$$

For example, with $\varphi = 2$ alternate samples are skipped, the wavetable sample rate halved and the output frequency precisely doubled, equivalent to an upward pitch shift of 1 octave.

For $N_c = 1$ (i.e. $f_a = \frac{f_s}{L}$) the wavetable is defined by $\mathbf{T}[a] = \cos\left(2\pi \frac{a}{L}\right), \quad a \in [0, L-1]$.

The resampled output sequence, $y(n)$, is then given by:

$$y(n) = \mathbf{T}\left[\langle n\varphi \rangle_L\right] = \cos\left(\frac{2\pi}{L}\langle n\varphi \rangle_L\right) = \cos\left(\frac{2\pi}{L}\left\langle n\frac{f_s}{f_s'}\right\rangle_L\right) \tag{4.1.12}$$

and illustrates the inverse relationship between *effective* wavetable sample rate, f_s', and the frequency of $y(n)$. Resampling a sequence $x(n) = \cos\left(2\pi \frac{f_a}{f_s} n\right)$ at a sample rate defined by Eq. (4.1.11) produces the sequence, $y(n) = \cos\left(2\pi \frac{f_a}{f_s'} n\right) = \cos\left(2\pi \frac{\varphi f_a}{f_s} n\right)$, which is equivalent to the frequency of $x(n)$ being scaled by φ.

We define the frequency control resolution, f_r, as the frequency change corresponding to a unit change in φ, thus:

$$f_r = \frac{f_s}{L} \tag{4.1.13}$$

and observe f_r can be reduced (i.e. improved) only by increasing L or decreasing f_s. However, decreasing f_s reduces the Nyquist frequency and hence maximum oscillation frequency in direct proportion. Increasing L increases memory cost and wavetable computation and loading time. We consider frequency resolution pertinent to computer

music requirements in section (4.2.5) and accept *a priori* that $L = 2^k$ where $k = O(24)$ for sufficiently precise frequency control in these applications.

4.1.4 Fractional Addressing

Frequency resolution is enhanced by employing the concept of *fractional addressing*, introduced by Max Mathews in his pioneering work at Bell Telephone Laboratories developing software based music synthesis oscillators [Mathews, 1969]. To understand fractional addressing, we first generalise the wavetable address counter as a phase accumulator with phase increment, φ. The wavetable address is now considered a time-varying phase argument. Fractional addressing of a tabulated signal requires extension of the phase accumulator resolution below the least significant address bit of the wavetable. We now have an *M*-bit fractional phase value in fixed-point format, partitioned into *I integer* bits, which address a radix-2 wavetable of length $L = 2^I$, and *F fraction* bits which represent the fractional address *between* adjacent wavetable entries. The phase increment, φ, is similarly partitioned into integer and fraction fields in a fixed-point format with an implied binary point between them as depicted in Figure (4.1.5). φ can now take on fractional values on the interval $\varphi \in [0, 2^I - 1]$ with resolution 2^{-F}. The phase resolution is thereby increased exponentially with each additional fraction bit. We investigate fractional addressing further in Chapter 5 and for our present discussion we accept that phase resolution can be improved by adding a fraction component to the phase accumulation computation.

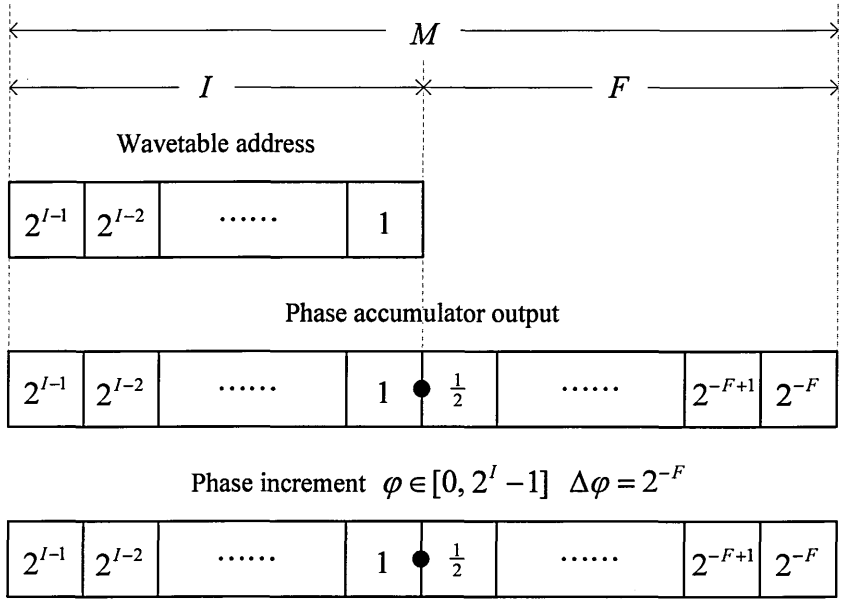


Figure (4.1.5): Data fields for fractional phase accumulation and wavetable addressing. (In rudimentary embodiments the phase word fraction component is discarded.)

Figures (4.1.6) and (4.1.7) illustrate examples of resampling a tabulated sinusoid with $\varphi = 2$ and $\varphi = 0.5$, respectively. For the case when $\varphi = 0.5$ (and all non-integer values of φ) the wavetable is being fractionally addressed. When $\varphi > 1$ we are reducing the wavetable sample rate (down-sampling) and increasing the synthesised frequency. Conversely, with $\varphi < 1$ we are increasing the wavetable sample rate (up-sampling) and decreasing the synthesised frequency. We can therefore model fractional address WLS as a *sample rate conversion* of the tabulated signal.

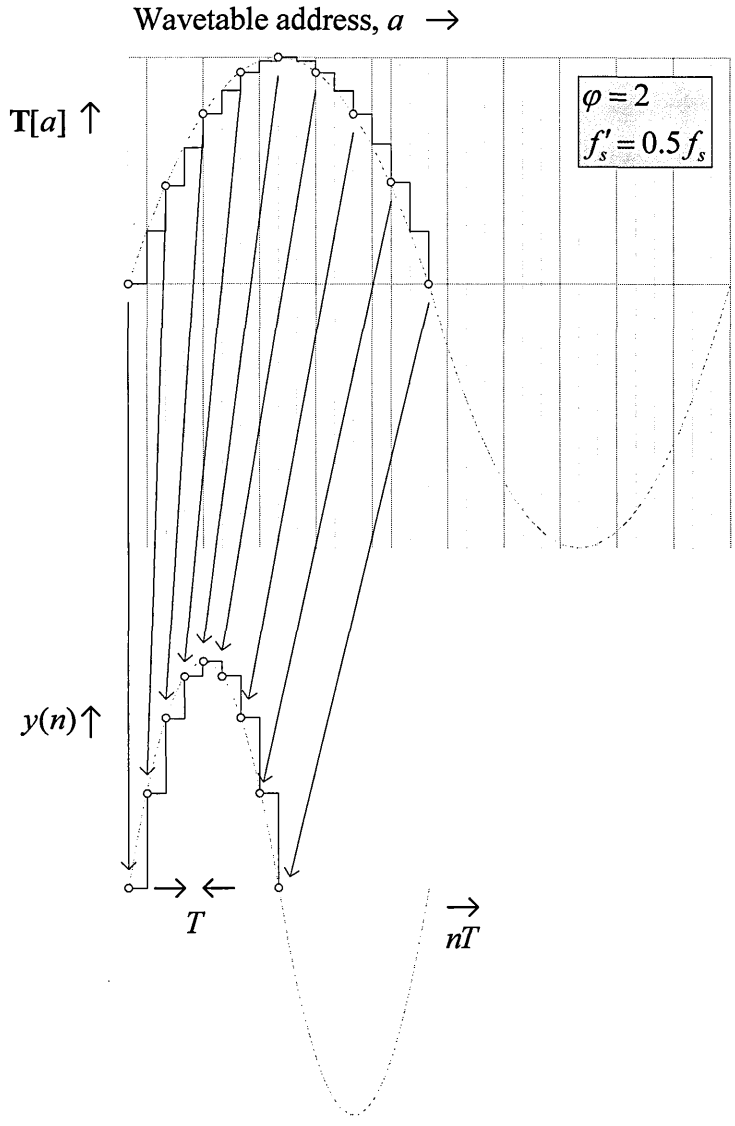


Figure (4.1.6): Time domain view of down-sampling a tabulated sinusoid with $\varphi = 2$ *and thereby increasing the frequency of* $y(n)$.

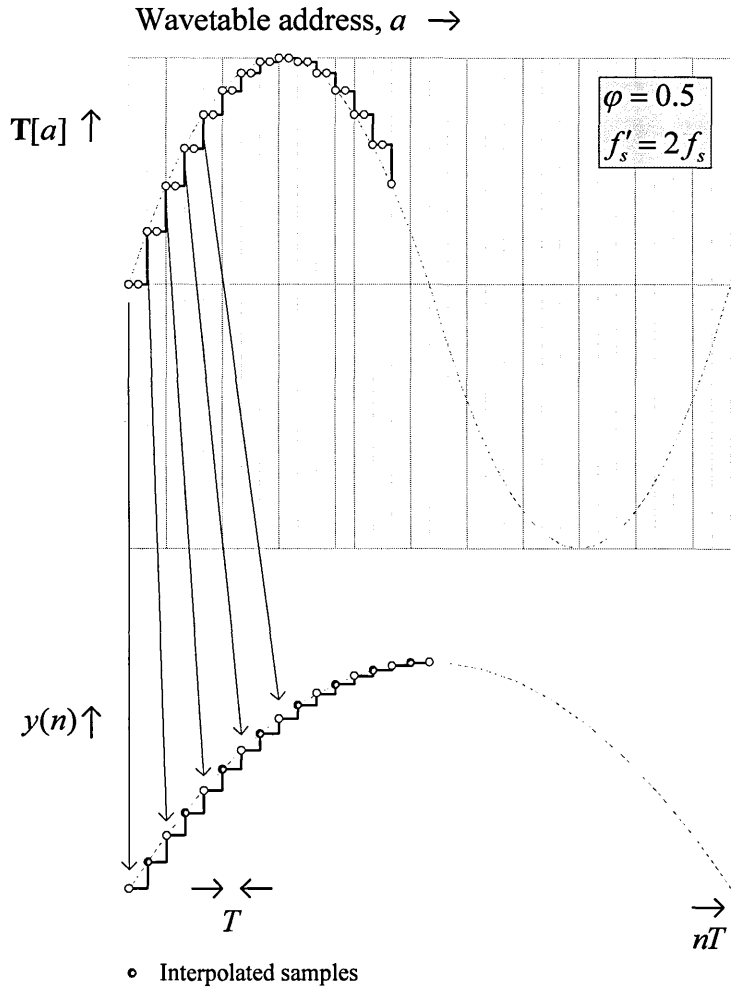


Figure (4.1.7): Time domain view of up-sampling a tabulated sinusoid with $\varphi = 0.5$ *and thereby decreasing the frequency of* $y(n)$. *The wavetable is being fractionally addressed and interpolated samples must be computed.*

4.1.5 The Sample-Rate-Conversion View

The process of generalised sample rate conversion is well documented in the literature [Proakis and Manolakis, 1996; Smith and Gossett, 1984 and Crochiere and Rabiner, 1983]. Here we present an overview of the salient points peculiar to WLS, which can be viewed as a *block processing* embodiment of the conversion algorithm. The classical sample rate conversion model of a sequence, $x(n)$, to a corresponding sequence, $y(m)$, requires that the conversion ratio be expressed as a reduced fraction, $\frac{U}{D}$, where the numerator and denominator have no common factors apart from 1. Sample rate conversion is then a three stage process – up-sample $x(n)$ by U, low-pass filter by $h(n)$ and down-sample by D as depicted in Figure (4.1.8).

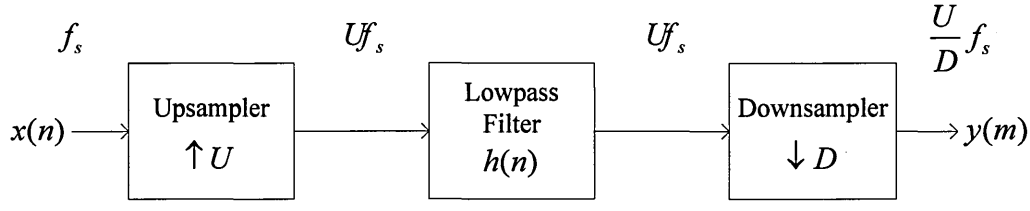


Figure (4.1.8): An analytical view of the sample rate conversion process.

We know from Eq. (4.1.11) that $\frac{f_s'}{f_s} = \frac{1}{\varphi} = \frac{U}{D}$, hence:

$$\varphi = \frac{D}{U} \tag{4.1.14}$$

Up-sampling is effected by inserting $U-1$ zero-valued samples in between the existing $x(n)$ samples, giving a sequence with sample rate Uf_s. Down-sampling is effected by selecting every D^{th} sample of the sequence. The low-pass filter, $h(n)$, has cutoff

frequency, f_c, whose value normalised to the input Nyquist frequency is given by $\hat{f}_c \leq \min\left(1, \frac{U}{D}\right)$ [Proakis and Manolakis, 1996]. Since $\varphi = \frac{D}{U}$ we have:

$$\hat{f}_c \leq \min\left(1, \frac{1}{\varphi}\right) \tag{4.1.15}$$

If we are decreasing the wavetable sample rate and increasing the frequency, we have $\hat{f}_c \leq \frac{1}{\varphi}$ and so $f_c \leq \frac{f_s}{2\varphi}$. Conversely, if we are increasing the wavetable sample rate and decreasing the frequency, we have $\hat{f}_c \leq 1$ and so $f_c \leq \frac{f_s}{2}$. We note that $h(n)$ is time-variant since φ and therefore $\frac{U}{D}$ is time variant. In its simplest form $h(n)$ is defined by the zero-order hold function $h(n+m) = x(n)$ for $m \in [0, U-1]$ [Massie, 1998]. This definition repeats the $x(n)$ sample $U-1$ times as depicted in Figure (4.1.7) instead of inserting $U-1$ zeros between samples as typically described in the literature. We see that $h(n)$ interpolates values between tabulated samples and will be a function of the fractional component of the DT phase signal. Figures (4.1.9) and (4.1.10) illustrate frequency domain examples of this process and indicate the upper bound on f_c.

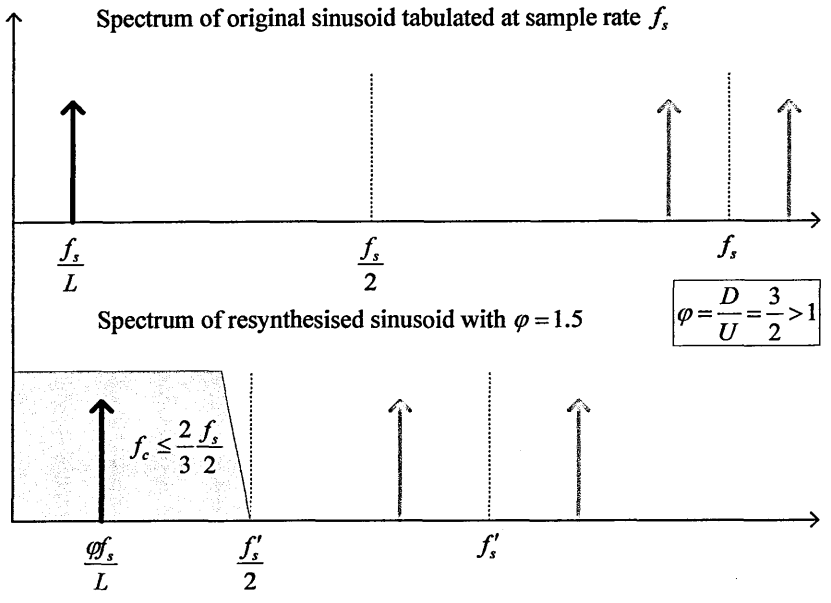


Figure (4.1.9): Frequency domain view of wavetable resampling with $\varphi = \frac{3}{2}$. *The* $h(n)$ *passband is shown shaded and sets an upper bound on* f_c.

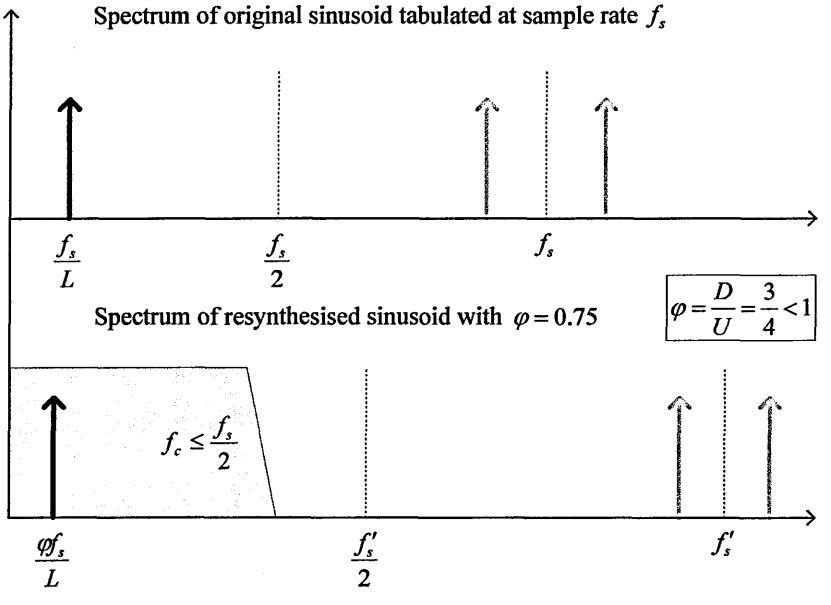


Figure (4.1.10): Frequency domain view of wavetable resampling with $\varphi = \frac{3}{4}$. *The* $h(n)$ *passband is shown shaded and sets an upper bound on* f_c. *(For* $\varphi < 1$ f_c *can remain fixed at* $f_c \leq \frac{f_s}{2}$.*)*

The sample rate conversion view of fractional address WLS is particularly appropriate when the wavetable contains a band-limited, non-sinusoidal signal. For the down-sampling condition when $\varphi > 1$, frequency components in the tabulated signal above the frequency, $\frac{f_s}{2\varphi}$, will alias into the Nyquist interval $[0, \frac{f_s}{2}]$. This condition is prevented by setting $f_c \leq \frac{f_s}{2\varphi}$. For the up-sampling condition when $\varphi < 1$ f_c can be fixed at $f_c \leq \frac{f_s}{2}$. In general, there are two tabulated signal classes to consider – sampled natural sounds comprising numerous cycles of the fundamental frequency and single-cycle periodic signals. We discuss the *sampling synthesis* model in section (4.2.6) where wavetables contain samples of natural instrument sounds.

Resampling wavetables containing a single cycle, band-limited periodic function causes components above $\frac{f_s'}{2}$ to alias into the Nyquist interval, $[0, \frac{f_s'}{2}]$, when the "new" Nyquist frequency, $\frac{f_s'}{2}$, falls below the highest frequency component in the wavetable signal. For a periodic wavetable function defined according to Eq. (4.1.3), the highest frequency component is simply the product of fundamental frequency and highest harmonic number, N_h, and so $N_h f < \frac{f_s}{2}$.

We know from Eq. (4.1.10) that $f = \frac{\varphi f_s}{L}$ and so we define an upper bound on φ, thus:

$$\varphi_{\max} = \frac{L}{2N_h} \tag{4.1.16}$$

and for $\varphi > \varphi_{\max}$ upper harmonics will alias into the Nyquist region.

4.2 Frequency Control

The preceding discussion presented an overview of wavetable lookup synthesis and the underlying principles for resynthesising at arbitrary frequencies. We now present a detailed review of phase accumulating frequency synthesis from first principles and its application to wavetable lookup synthesis. The simple integer phase increment oscillator discussed earlier requires a large wavetable to achieve the frequency control resolution necessary for computer music applications. The concept of fractional addressing enables reduced wavetable length simultaneous with arbitrarily high frequency control resolution. The penalty for decoupling wavetable length and frequency control resolution is the introduction of spectrally complex noise components into the output signal that can only be reduced by interpolating the phase-amplitude mapping process and incurring additional computational overhead.

4.2.1 The Phase-Frequency Relationship in DT Sinusoid Synthesis

Frequency may be defined as the number of complete cycles of a periodic process (e.g. a sinusoid) occurring per unit time. Interpretations of frequency in the DT domain are different compared to the CT case. For example, the DT phasor $y(n) = e^{j\omega n} = \cos\omega n + j\sin\omega n$, with angular frequency, ω, and integer indexing variable, $n \in (-\infty, \infty)$, is periodic if and only if the frequency (expressed in radians per sample) is a *rational* multiple of 2π. That is, the frequency is the ratio of two integers.

We now review the continuous-time (CT) sinusoid as a progenitor for the DT case through the sampling process. A CT sinusoid may be expressed as $y_a(t) = A\cos(\Omega t + \theta)$, $t \in (-\infty, \infty)$, where A is the amplitude of the sinusoid, Ω is the angular frequency in radians per second, θ is the phase in radians and t is time. We can also consider the frequency, F, in cycles per second (Hz) since $\Omega \equiv 2\pi F$ and so

$y_a(t) = A\cos(2\pi Ft + \theta)$, $t \in (-\infty, \infty)$. We see that for every constant value of F, $y_a(t)$ is periodic and so $y_a(t+\tau) = y_a(t)$, where the period, τ, is given by $\tau = \frac{1}{F}$. CT sinusoids with distinct frequencies are themselves distinct. Increasing the frequency increases the rate of oscillation and produces more cycles within a given time interval. The instantaneous phase, $\phi(t)$ and angular frequency, $\omega(t)$, of a CT sinusoid are interrelated according to the integral and differential equations:

$$\phi(t) = \int_0^t \omega(t)dt + C$$
$$\omega(t) = \frac{d\phi(t)}{dt} \qquad (4.2.1)$$

In general, we set $C = \phi(0) = \theta$ and hence the instantaneous phase for constant $\omega(t)$ is in precise agreement with the argument of a generalised CT sinusoid (i.e. $\phi(t) = \omega t + \theta$).

A discrete-time (DT) sinusoid may be expressed as $y(n) = A\cos(\omega n + \theta)$, $n \in (-\infty, \infty)$, where A is the amplitude of the sinusoid, ω is the angular frequency in radians per sample, θ is the phase in radians and n is the sample index which takes on integer values. Since $\omega = 2\pi f$, we also have $y(n) = A\cos(2\pi f n + \theta)$, $n \in (-\infty, \infty)$, where f represents the frequency of the DT sinusoid in cycles per sample. DT sinusoids possess three important properties:

- A DT sinusoid is periodic only if its frequency, f, is a rational number. The smallest value of N for which $y(n+N) = y(n)$ is the *fundamental period* [Proakis & Manolakis, 1996]. So for a DT sinusoid to be periodic we require $\cos(2\pi f(n+N) + \theta) = \cos(2\pi fn + \theta)$ which is only satisfied if there exists an

integer, k, such that $2\pi fN = 2k\pi$ or $f = \frac{k}{N}$. Since f is the ratio of two integers, f must be a rational number.

- DT sinusoids whose angular frequencies are separated by an integer multiple of 2π are identical and indistinguishable from each other since $\cos[(\omega + 2k\pi)n + \theta] = \cos(\omega n + 2k\pi n + \theta) = \cos(\omega n + \theta)$ for all integer values of k.
- The highest frequency of a DT sinusoid is attained when $\omega = \pm\pi$, or equivalently when $f = \pm\frac{1}{2}$.

The sample index, n, counts discrete time intervals whose spacing is the sample period. Sampling the CT signal $y_a(t) = A\cos(2\pi Ft + \theta)$ with sample rate $f_s = \frac{1}{T}$ establishes the relationship $t = nT$ between the CT and DT domains and yields the DT sinusoid $y_a(nT) \equiv y(n) = A\cos(2\pi FnT + \theta)$. Comparing this expression with the *generalised* DT sinusoid $y(n) = A\cos(2\pi fn + \theta)$, we determine that $f = \frac{F}{f_s}$ which denotes the *relative* or *normalised frequency* of the sinusoid [Proakis and Manolakis, 1996]. Therefore, the interrelationship between f and F can only be defined if the sampling frequency is known. For $F \in [-\frac{f_s}{2}, \frac{f_s}{2}]$ then $f \in [-\frac{1}{2}, \frac{1}{2}]$ and so the relationship between f and F is one-one and we can reconstruct the CT (analogue) signal, $y_a(t)$, from the sequence, $y(n)$. In general, we may express a DT sinusoid thus:

$$y(n) = \cos(2\pi FnT + \theta) \qquad (4.2.2)$$

where $F \in [-\frac{f_s}{2}, \frac{f_s}{2}]$.

For DT sinusoids with constant frequency (i.e. $\frac{d\omega(t)}{dt}=0$), the underlying phase sequence, $\phi(n)$, is a sawtooth function with constant slope (ω) and maximum amplitude 2π. For the constant frequency condition we may define $\phi(n)$ exactly by the difference equation:

$$\phi(n)=T\frac{d\phi(n)}{dt}+\phi(n-1) \qquad (4.2.3)$$

where $\frac{d\phi(n)}{dt}=\omega$. If we set the initial condition $\phi(-1)=\theta-\omega T$, then the general solution is simply $\phi(n)=(n\omega T+\theta)=(2\pi fnT+\theta)$, which is in precise agreement with the argument of a DT sinusoid.

$\phi(n)$ is computed by a modulo 2π accumulation of a phase increment, φ, at the sample rate, f_s, and can therefore be expressed by the difference equation:

$$\phi(n)=\langle\phi(n-1)+\varphi\rangle_{2\pi} \qquad (4.2.4)$$

where the frequency of the $\phi(n)$ sequence is a function of φ. In a physical realisation, $\phi(n)$ and φ are represented by unsigned binary numbers of width M bits. Modulo operation is effected by discarding the carry in a binary accumulation comprising an M-bit adder and register as depicted in Figure 4.2.1. The accumulation of φ proceeds modulo 2^M as the accumulator overflows. The *radian* value of the phase sequence, $\phi_r(n)$, is given by $\phi_r(n)=\phi_M(n)\frac{2\pi}{2^M}$, where $\phi_M(n)\in[0,2^M-1]$ and represents the phase value from an M-bit integer accumulator. By tabulating a single cycle sinusoid with phase increment $\frac{2\pi}{2^M}$ and $L=2^M$ samples as in Eq. (4.1.2), the address interval $[0,2^M)$ corresponds to the phase interval $[0,2\pi)$.

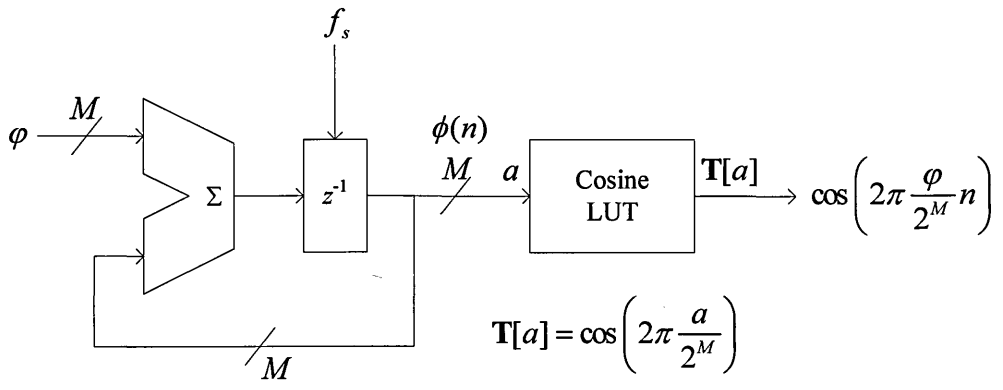


Figure (4.2.1): An M-bit accumulator with phase increment, φ, generates an M-bit phase sequence, $\phi(n)$, which addresses a 2^M location wavetable to effect phase-amplitude mapping.

The phase accumulator may be considered as a DT integrator followed by a modulo 2^M operator computing a modulo 2^M integration of phase increment to give frequency. Using Eqs. (4.2.1) and (4.2.3) we express the output frequency, f, as a function of M, φ and f_s, thus:

$$f = \frac{1}{2\pi}\frac{\phi(n)-\phi(n-1)}{T} = \frac{\varphi}{2^M T} = \frac{\varphi f_s}{2^M}$$
$$\varphi \in [1, 2^{M-1}-1] \tag{4.2.5}$$

Eq. (4.2.5) is fundamental to phase accumulating WLS and indicates two important properties – f is directly proportional to φ and frequency resolution as defined by $\frac{df}{d\varphi}$ is precisely $\frac{f_s}{2^M}$.

4.2.2 Frequency Control Precision

Human pitch perception is based on the *ratio* of frequencies rather than absolute frequency [Moore, 1990]. The equal tempered scale defines the *half-step* or *semitone* frequency ratio as $\sqrt[12]{2} \approx 1.059463$. Since this tuning system is relative, a basis

frequency is needed to define an absolute pitch scale. For the equal tempered scale, this basis frequency is defined by the international pitch standard [Moore, 1990] as *precisely* 440 Hz and defines the A above middle-C (A4) on a conventionally tuned piano. The *cent* is defined as one-hundredth of a half-step or a ratio of $\sqrt[1200]{2} \approx 1.000578$. In general, a $\pm k$ cent pitch shift equates to frequency scaling of $\left(\sqrt[1200]{2}\right)^{\pm k}$.

There is a limit to human audio perception in distinguishing the relative pitch between two sinusoidal tones of equal intensity presented one after the other. When the difference in frequency between the two tones is below this limit, both tones are perceived as having the same pitch. When the frequency difference exceeds the *just noticeable difference* (JND) threshold a change in pitch is perceived. The pitch change JND is widely reported in the literature and depends on the frequency, intensity and duration of the test tone. It also varies greatly from person to person, is affected by the level of musical training and depends considerably on the measurement technique employed [Roederer, 1973, Rossing, 1990]. The literature consensus is that a 5 cent pitch change represents the *order* of JND for most listeners and listening conditions, although Rakowski [1971] reports that under ideal listening conditions a very well trained ear can discriminate a frequency change of between 0.03 and 0.08 Hz around 160 Hz. This corresponds to a pitch change JND of approximately 0.3 cents. The pitch change JND is crucial in determining the *absolute* tuning accuracy or frequency resolution of a musical oscillator. However, frequency resolution *below* that required to satisfy the pitch change JND is needed when the *beat frequency* between two (or more) closely tuned oscillators must be precisely controlled – a common requirement in orchestral computer music synthesis. A 5 cent pitch tuning resolution at 7040 Hz (A8) requires a frequency resolution of 20.4 Hz which is inadequate for precise beat

frequency control (i.e. beat frequencies could only be specified in integer multiples of 20.4 Hz).

We now consider phase accumulator frequency resolution taking into account both pitch change JND and beat frequency tuning across the audio range. We postulate that a frequency resolution on the order of 0.01 Hz will provide sufficient beat frequency control resolution for all conceivable musical applications. Furthermore, we consider the impact of achieving a pitch tuning accuracy between 0.3 and 5 cents. The number of phase accumulator bits, M, is related to the frequency resolution, f_r, according to:

$$M = \left\lceil \frac{1}{\log 2} \log\left(\frac{f_s}{f_r}\right) \right\rceil \qquad (4.2.6)$$

For a sample rate of 48 kHz, a frequency resolution of 0.01 Hz requires $M = 23$. Since a pitch change is a fixed ratio of frequencies, a given frequency resolution will represent an increasing pitch tuning error with reducing frequency. We define the *pitch tuning error*, ε_p, in cents, as the frequency *ratio* corresponding to a unit change in φ for a given minimum oscillation frequency, $f_{\min}$, sampling rate, f_s and phase accumulator word size, M. We have:

$$\varepsilon_p = \frac{1200}{\log 2} \log\left(1 + \frac{f_s}{2^M f_{\min}}\right) \qquad (4.2.7)$$

The pitch tuning error is plotted in Figure (4.2.2) as a function of M with $f_{\min} = 27.5$ Hz (A0) and three values of f_s – 48 kHz, 96 kHz and 192 kHz, with 5 cent and 0.3 cent markers shown for reference. Pitch tuning error is maintained below 1 cent at all frequencies above 27.5 Hz when $M \geq 22$ for $f_s = 48$ kHz and $M \geq 24$ for $f_s = 192$ kHz. Figure (4.2.3) illustrates the variation of frequency resolution, $f_r = \frac{f_s}{2^M}$, with M for the same three values of f_s.

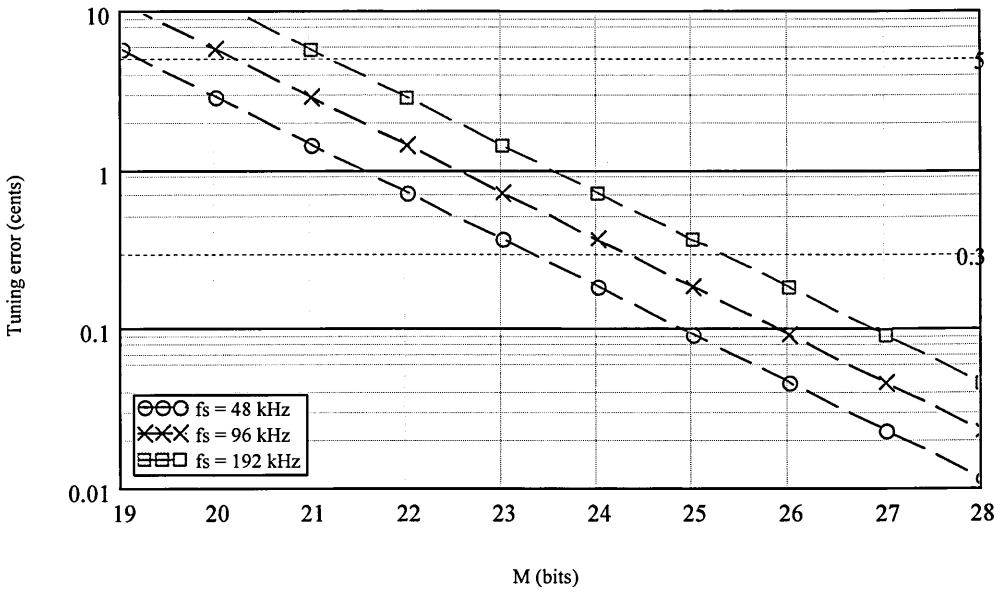


Figure (4.2.2): Variation of pitch tuning error, ε_p, with M for $f_{\min} = 27.5\text{Hz}$.

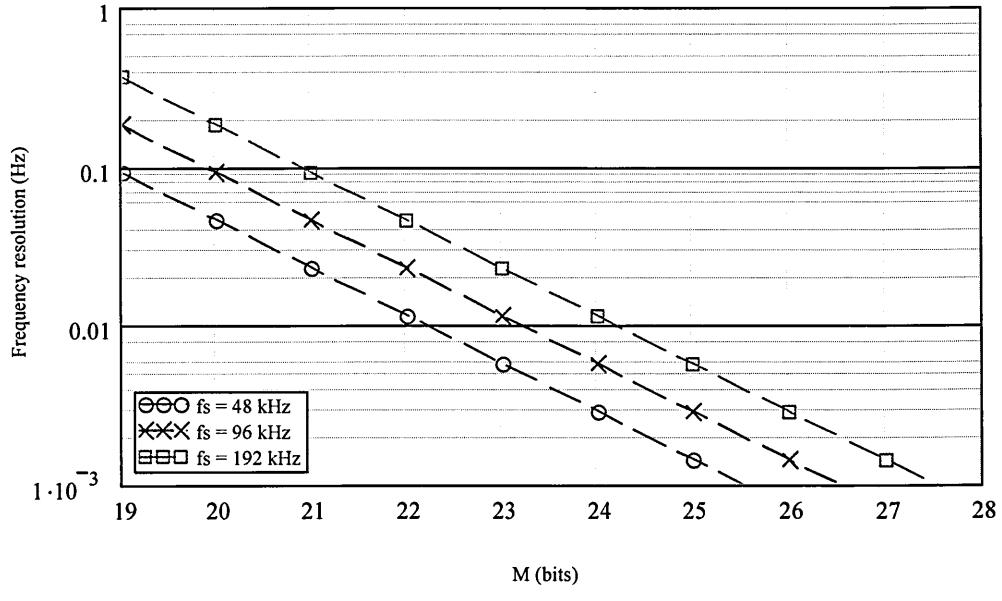


Figure (4.2.3): Variation of frequency resolution with M for three values of f_s.

We conclude from inspection of Figures (4.2.2) and (4.2.3) that acceptable M values lie in the range 20 to 24 depending on JND threshold and sampling frequency. Choosing $M = 24$ provides surety of pitch and beat frequency tuning accuracy for all f_s values peculiar to computer music applications. Increasing the sampling frequency over that required for normal audio bandwidth (oversampling) causes a corresponding increase in the Nyquist frequency reducing susceptibility to upper harmonics aliasing when complex wavetables are resynthesised.

4.2.3 Phase Accumulation and Phase Continuity

For modulo 2^M phase accumulation, Eq. (4.2.4) becomes $\phi(n) = \langle \phi(n-1) + \varphi \rangle_{2^M}$ and in general, the $\phi(n)$ sequence with initial condition, $\phi(0)$, is given by:

$$\phi(n) = \left\langle \phi(0) + \sum_{i=1}^{n} \varphi \right\rangle_{2^M} = \langle \phi(0) + n\varphi \rangle_{2^M} \qquad (4.2.8)$$

For an M-bit unsigned binary accumulator there are 2^M possible phase states represented by the set $\Phi = \{0,1,2,\ldots(2^M - 1)\}$ and so $\phi(n) \in \Phi$. The frequency of the phase sequence is given by Eq. (4.2.5) where the upper limit on the range of φ is imposed by the Nyquist sampling theorem to ensure $f < \frac{f_s}{2}$. Values of φ greater than 2^{M-1} produce an aliased output frequency equal to $f_s - f$.

Figure (4.2.4) illustrates an example phase sequence given by $\phi(n) = \langle 7n \rangle_{32}$.

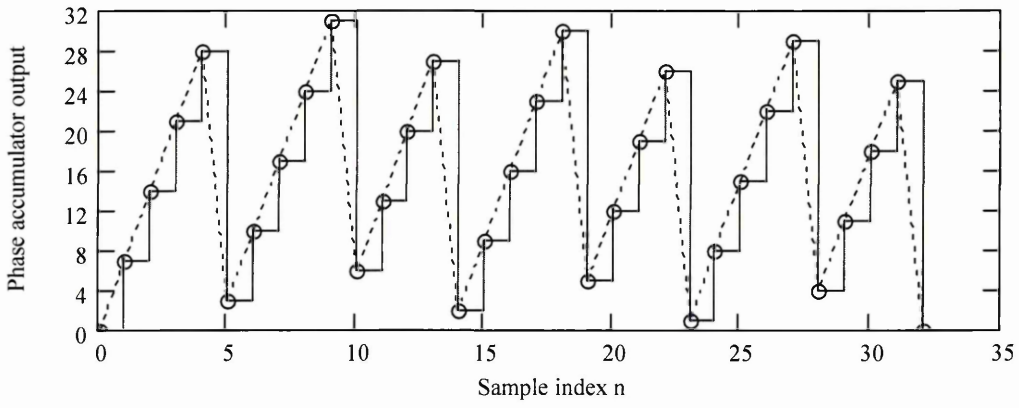


Figure (4.2.4): Phase accumulator output sequence with $M = 5$ and $\varphi = 7$.

It is observed that the phase sequence comprises different values on successive cycles and repeats every 32 samples. The *average* period of the underlying sawtooth function computed over the numerical period of this sequence is in *precise* agreement with Eq. (4.2.5), as illustrated in Table (4.2.1).

$\phi(n)$ **Segment**	**Period**	$\phi(n)$ **Segment**	**Period**
$\{0,7,14,21,28\}$	5	$\{5,12,19,26\}$	4
$\{3,10,17,24,31\}$	5	$\{1,8,15,22,29\}$	5
$\{6,13,20,27\}$	4	$\{4,11,18,25\}$	4
$\{2,9,16,23,30\}$	5	*Average* period	$\frac{32}{7} = \frac{2^M}{\varphi}$

Table (4.2.1): Illustrating the precise average period of the sawtooth phase sequence $\langle 7n \rangle_{32}$.

In Chapter 3 we discussed phase continuity and the corresponding amplitude domain effects of phase discontinuous transitions. Phase continuous frequency transition is an

essential attribute in a digital oscillator, particularly in computer music applications. Before considering phase continuity in $\phi(n)$ about a change in φ, we consider a step change in the frequency parameter of a DT sinusoid at an arbitrary sample index. We first define a step change in the frequency parameter, $f(n)$, at sample index m, thus:

$$f(n) = \begin{cases} f & n < m \\ f' & n \geq m \end{cases} \tag{4.2.9}$$

It is apparent by considering the DT sinusoid $y(n) = A\cos(2\pi f(n)nT + \theta), -\infty < n < \infty$ that a step change in $f(n)$ at $n = m$ produces a corresponding step change in the sine function argument, $\psi(n) = 2\pi f(n)nT + \theta$. This causes a discontinuity in $y(n)$ as depicted in Figure (4.2.5a) with the underlying phase sequence shown in Figure (4.2.5b). The magnitude of the step change in phase is $2\pi T(mf' - (m-1)f)$ radians.

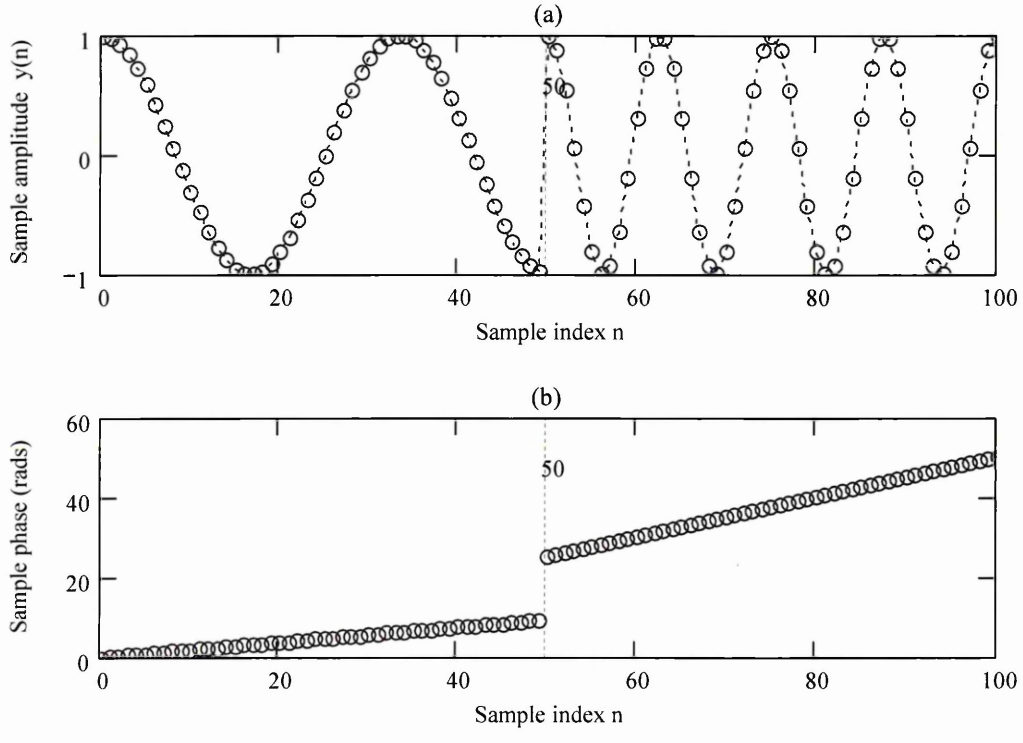


Figure (4.2.5): (a) - A DT sinusoid with phase discontinuous frequency transition at $n = 50$. *(b) - Corresponding phase sequence.*

A phase continuous frequency change in $y(n)$ requires that we compute the underlying phase function, $\psi(n)$, by integrating $f(n)$ over n samples. Since this is a DT system, this integration can be represented by the two stage summation assuming $n \geq m$, thus:

$$\psi(n)\big|_{n\geq m} = 2\pi\sum_{0}^{nT} f(n) = 2\pi\sum_{0}^{mT} f + 2\pi\sum_{mT}^{nT} f' = 2\pi f'nT - 2\pi(f'-f)mT \tag{4.2.10}$$

In general, we have:

$$\psi(n) = \begin{cases} 2\pi fnT & n < m \\ 2\pi f'nT - 2\pi(f'-f)mT & n \geq m \end{cases} \tag{4.2.11}$$

Eq. (4.2.11) provides an analytical definition of a phase continuous sequence for $n \geq m$, effectively subtracting the phase discontinuous step change from the $2\pi f'nT$ phase argument. The amount subtracted is the magnitude of the step change, $2\pi T[mf'-(m-1)f]$, less the phase increment at the original frequency, $2\pi fT$, as illustrated in Figure (4.2.6). Eq. (4.2.11) defines a generalised phase sequence, $\psi(n)$, having an instantaneous change in slope at the frequency transition point with no step discontinuity, thereby producing a phase continuous change in $y(n)$ as illustrated in Figure (4.2.7).

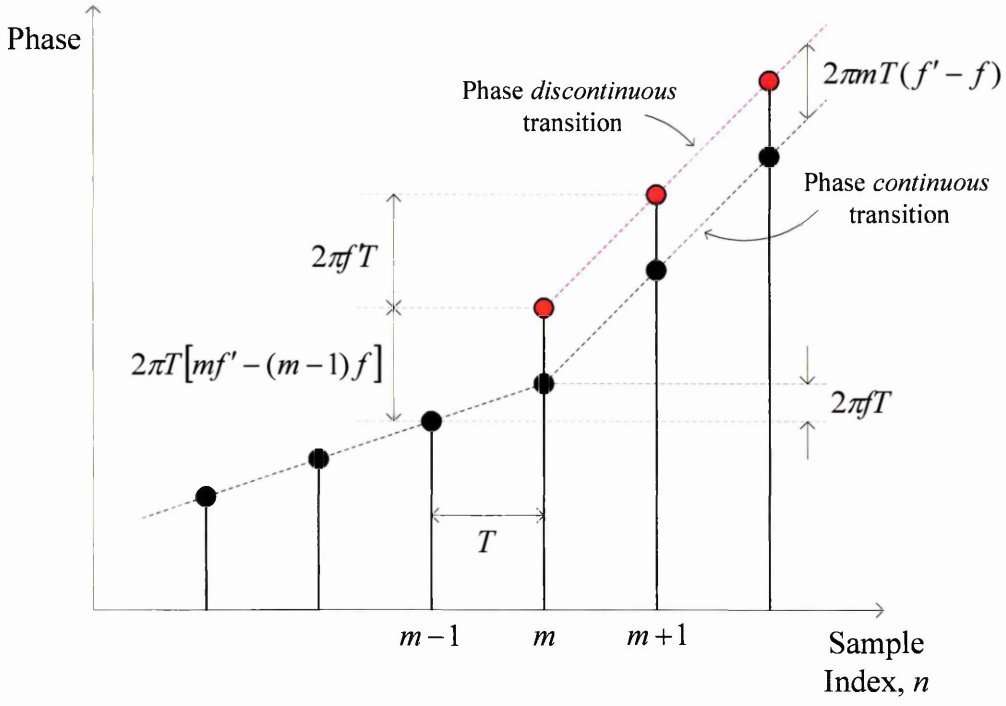


Figure (4.2.6): Graphical representation of phase-discontinuous and continuous sequences. Frequency transition from f to f' occurs at sample index m. The phase continuous sequence is defined by Eq. (4.2.11).

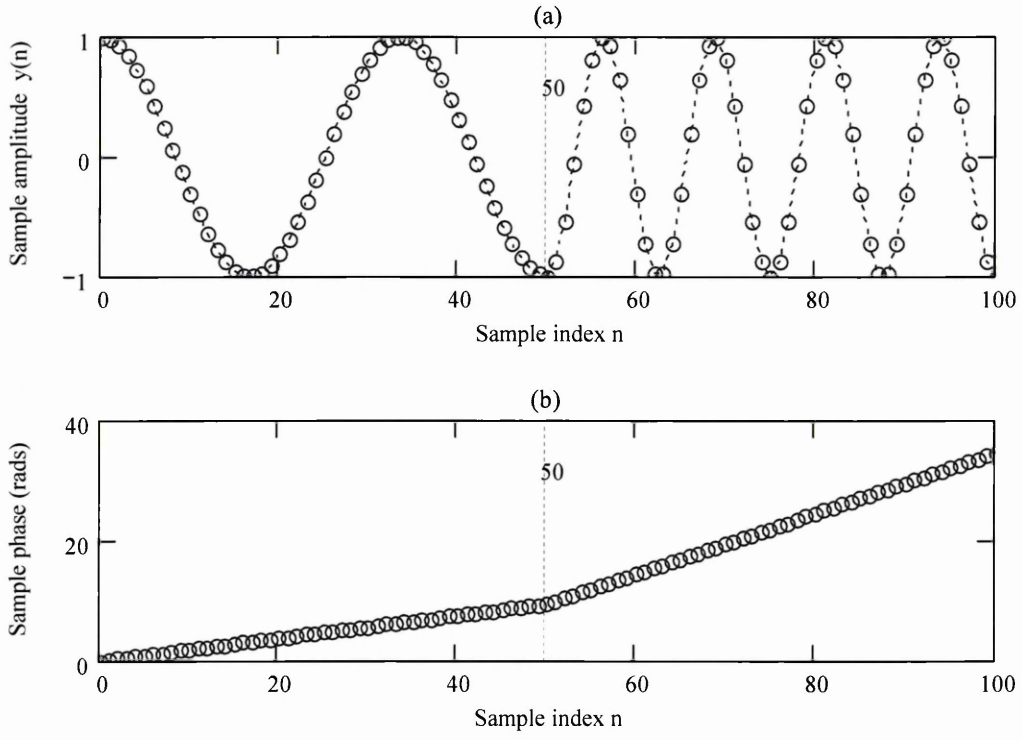


Figure (4.2.7): (a) - A DT sinusoid with phase continuous frequency transition at $n = 50$. *(b) - Corresponding phase sequence.*

Let us now consider a step change in the phase accumulator increment parameter at sample m, given by:

$$\varphi(n) = \begin{cases} \varphi & n < m \\ \varphi' & n \geq m \end{cases} \tag{4.2.12}$$

Following similar reasoning to the development of Eqs. (4.2.10) and (4.2.11), we obtain:

$$\phi(n)\big|_{n\geq m} = \left\langle \sum_{0}^{m} \varphi \right\rangle_{2^M} + \left\langle \sum_{m}^{n} \varphi' \right\rangle_{2^M} = \left\langle n\varphi' - m(\varphi' - \varphi) \right\rangle_{2^M} \tag{4.2.13}$$

and hence:

$$\phi(n) = \begin{cases} \langle n\varphi \rangle_{2^M} & n < m \\ \langle n\varphi' - m(\varphi' - \varphi) \rangle_{2^M} & n \geq m \end{cases} \tag{4.2.14}$$

Since Eq. (4.2.14) is precisely analogous to Eq. (4.2.11) we conclude that the phase accumulator produces phase continuous output sequences following a step transition in φ at *any sample index.* Figure (4.2.8) illustrates the inherently phase-continuous output sequence of a phase accumulator with step change in phase increment, φ, at sample index $m = 50$. The corresponding phase mapped sinusoidal signal is also shown.

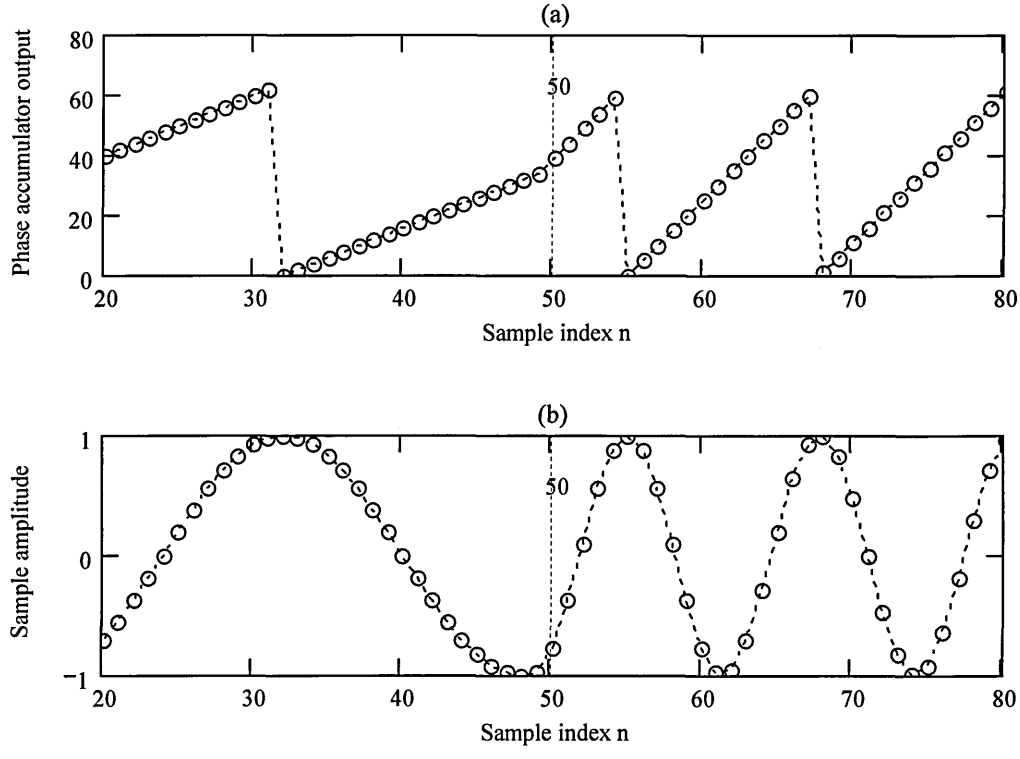


Figure (4.2.8): A typical phase accumulator output sequence (a) and associated phase mapped sinusoidal sequence (b), with $M = 6$, $\varphi = 2$, $\varphi' = 5$ *and* $m = 50$.

The frequency of the phase sequence, $\phi(n)$, is a minimum when $\varphi = 1$ when all phase states are output contiguously, taking 2^M sample clocks to complete a cycle. The frequency resolution, f_r, is therefore given by:

$$f_r = \frac{f_s}{2^M} \tag{4.2.15}$$

and is governed only by M and f_s. The value of M (and therefore the frequency resolution) is constrained only by carry propagation time within a physical realisation of the accumulator adder and may be set at any desired value to achieve a specific frequency resolution consistent with hardware computational speed.

4.2.4 Optimal Phase Mapping

A wavetable maps a phase sequence to an arbitrary waveform amplitude sequence according to its tabulated values. This mapping will be optimal (i.e. utilise all *available* phase information) when all M bits of the phase sequence, $\phi(n)$, are used to address the wavetable. However, we have seen in section (4.2.2) that the typical values of M required in computer music applications prohibit this condition. To accommodate a reduced wavetable length, the M-bit phase sequence is truncated (or rounded) to I bits with $I < M$ and $L = 2^I$ [Moore, 1990]. The wavetable now comprises 2^I locations and the implicit phase truncation generates phase-amplitude mapping errors that manifest as distortion components in the output signal which we consider in Chapter 5.

We now consider the ideal case of phase mapping to a sinusoidal sequence with no phase truncation (i.e. $I = M$). We also assume that the tabulated samples are stored with infinite precision allowing us to ignore amplitude quantisation effects and focus on error components corresponding to phase mapping alone. A 2^M location sinusoidal wavetable, whose a^{th} element we represent by $\mathbf{C}[a]$, is defined by:

$$\mathbf{C}[a] = \cos\left(2\pi \frac{a}{2^M}\right)$$
$$a = 0,1,2,3,\ldots(2^M - 1) \qquad (4.2.16)$$

Since all phase sequence bits are used in the phase mapping we have $\phi(n) = \langle n\varphi \rangle_{2^M}$ and so the output sequence, $y(n)$, is given by:

$$y(n) = \mathbf{C}[\phi(n)] = \cos\left(2\pi \frac{\varphi}{2^M} n\right) \tag{4.2.17}$$

where $y(n)$ represents a DT sinusoid of frequency $f_0 = \frac{\varphi f_s}{2^M}$, with discrete Fourier transform, $\hat{Y}(f)$, given by the Poisson summation formula [Orfanidis, 1996]:

$$\begin{aligned} \hat{Y}(f) &= \sum_{n=-\infty}^{\infty} y(nT) e^{-2\pi j f n T} = \sum_{n=-\infty}^{\infty} \cos\left(2\pi \frac{\varphi}{2^M} n\right) e^{-2\pi j f n T} \\ &= \frac{1}{T} \sum_{n=-\infty}^{\infty} Y(f - n f_s) \end{aligned} \tag{4.2.18}$$

where $Y(f)$ represents the spectrum of the periodic CT signal $y(t) = \cos(2\pi f_0 t)$ with $f_0 = \frac{\varphi f_s}{2^M}$ and given by:

$$Y(f) = \sum_{k=-\infty}^{\infty} c_k \delta(f - k f_0) \tag{4.2.19}$$

where $\delta(n)$ is the unit impulse function defined as $\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & \text{otherwise} \end{cases}$. Since $y(t)$ is periodic with period $T_0 = \frac{1}{f_0}$, the Fourier coefficients are given by:

$$c_k = \frac{1}{T_0} \int_0^{T_0} y(t) e^{-2\pi j f_0 k t} dt = \begin{cases} \frac{1}{2} & k = \pm 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.2.20}$$

Hence the discrete spectrum of $y(n)$ is given by:

$$\hat{Y}(f) = f_s \sum_{n=-\infty}^{\infty} \sum_{k=-1}^{1} \delta(f - k f_0 - n f_s) \tag{4.2.21}$$

and represents all the frequencies in the periodically replicated set $\{(f_0 + k f_s),\ \ k = 0, \pm 1, \pm 2, ...\}$. We therefore conclude there are only alias images of f_0 in the spectrum of $y(n)$.

We now consider the output spectrum of a 2^M location wavetable, $\mathbf{W}$, containing a single cycle, periodic band-limited signal, whose a^{th} value we represent by $\mathbf{W}[a]$. $\mathbf{W}$ tabulates the weighted sum of N_h harmonics (a truncated Fourier series with zero DC term), thus:

$$\mathbf{W}[a] = \sum_{k=0}^{N_h} A_k \cos\left(2\pi k \frac{a}{2^M} + \Phi_k\right)$$

$$a = 0,1, \ldots (2^M - 1) \tag{4.2.22}$$

$$k = 0,1, \ldots N_h, \quad N_h \in [1, 2^{M-1} - 1]$$

where $A_k \in [0,1]$ and $\Phi_k \in [0, 2\pi)$ respectively denote the k^{th} harmonic amplitude and phase coefficients with DC term $A_0 = 0$. The wavetable is indexed by all M bits of the phase sequence giving the amplitude sequence, $y_w(n)$:

$$y_w(n) = \mathbf{W}[\phi(n)] = \sum_{k=0}^{N_h} A_k \cos\left(2\pi k n \frac{\varphi}{2^M} + \Phi_k\right) \tag{4.2.23}$$

Following a similar argument to the sinusoidal wavetable case, we have:

$$\begin{aligned} y_w(t) &= \sum_{k=0}^{N_h} A_k \cos(2\pi f_0 k t + \Phi_k) \\ &= \sum_{k=0}^{N_h} \left(\tfrac{1}{2} e^{j(2\pi f_0 k t + \Phi_k)} + \tfrac{1}{2} e^{-j(2\pi f_0 k t + \Phi_k)}\right) \end{aligned} \tag{4.2.24}$$

where $y_w(t)$ represents the CT signal sampled at f_s to give $y_w(nT)$ at sampling instants $t = nT$. The Fourier coefficients, c_l, are given by:

$$c_l = \frac{1}{T_0}\left(\frac{A_k}{2} e^{j\Phi_k} \int_0^{T_0} e^{2\pi j f_0 (k-l)t}\, dt + \frac{A_k}{2} e^{-j\Phi_k} \int_0^{T_0} e^{-2\pi j f_0 (k+l)t}\, dt \right)$$

$$= \begin{cases} \frac{A_k}{2} e^{j\Phi_k} & \text{for } \; l = k \\ \\ \frac{A_k}{2} e^{-j\Phi_k} & \text{for } \; l = -k \end{cases} \qquad (4.2.25)$$

$$k \in [0, N_h] \qquad l \in [0, N_h]$$

Substituting for c_k from Eq. (4.2.25) into Eq. (4.2.19) gives the corresponding spectrum of $y_w(t)$, thus:

$$Y_w(f) = \sum_{k=-N_h}^{N_h} A_k e^{j\Phi_k} \delta(f - kf_0) \qquad (4.2.26)$$

The spectrum, $\hat{Y}_w(f)$, is given by substituting for $Y_w(f)$ from Eq. (4.2.26) into Eq. (4.2.18), thus:

$$\hat{Y}(f) = f_s \sum_{n=-\infty}^{\infty} \sum_{k=-N_h}^{N_h} A_k e^{j\Phi_k} \delta(f - kf_0 - nf_s) \qquad (4.2.27)$$

Setting Φ to zero we have $c_k = \frac{A_k}{2}$ and so:

$$\hat{Y}(f) = \frac{f_s}{2} \sum_{n=-\infty}^{\infty} \sum_{k=-N_h}^{N_h} A_k \delta(f - kf_0 - nf_s) \qquad (4.2.28)$$

Eq. (4.2.28) shows that under the optimal phase mapping condition all harmonics of $y_w(nT)$ are aliased about *every* integer multiple of the sample frequency, f_s, with no other frequency components present.

For completeness, we consider the effects of the zero-order hold implicit on conversion into the CT domain. The zero-order hold has impulse response:

$$h(t) = \begin{cases} 1 & 0 \le t \le T \\ 0 & \text{otherwise} \end{cases} \qquad (4.2.29)$$

and therefore frequency response, $H(f)$, given by:

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{-j2\pi ft}\, dt = \int_{0}^{T} e^{-j2\pi ft}\, dt = T\left(\frac{\sin(\pi fT)}{\pi fT}\right) \tag{4.2.30}$$

Hence Eq. (4.2.28) becomes:

$$\hat{Y}(f) = \left(\frac{\sin(\pi fT)}{\pi fT}\right) \sum_{n=-\infty}^{\infty} \sum_{k=-N_h}^{N_h} A_k \delta(f - kf_0 - nf_s) \tag{4.2.31}$$

Figure (4.2.9) illustrates the spectrum $\hat{Y}(f)$ under *optimal* phase mapping conditions ($L = 2^M$), indicating that only the base spectrum and its alias images are present.

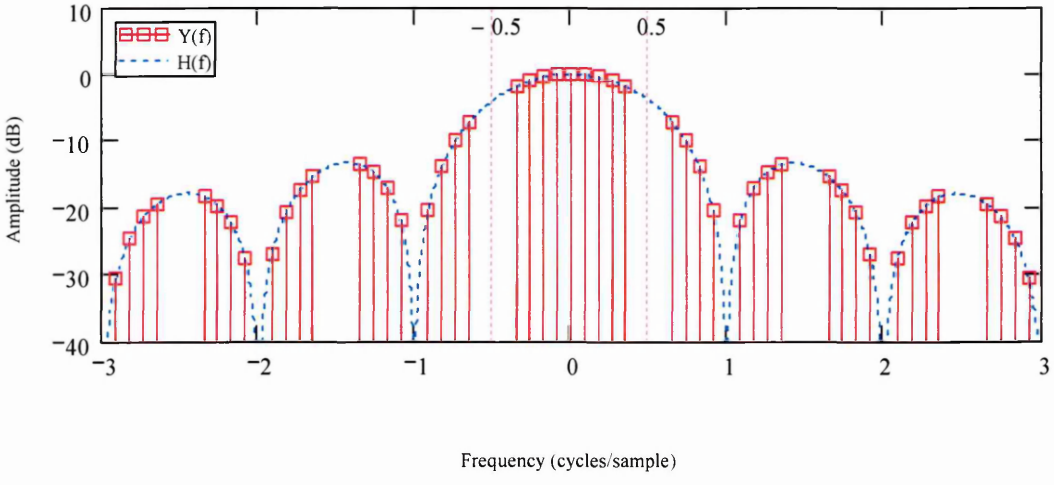


Figure (4.2.9): Resynthesised spectrum, $\hat{Y}(f)$, from a 4 harmonic wavetable with harmonic amplitudes set to unity to illustrate how alias images conform to the zero-order hold frequency response, $H(f)$.

We conclude that the output spectrum of the phase accumulating wavetable oscillator with *non-truncated* phase mapping is in precise agreement with the spectrum of an equivalent sampled continuous time signal for all frequencies up to the Nyquist limit.

4.2.5 Phase Control

The partitioned nature of phase accumulating WLS permits precise control of signal phase. Adding an offset $p \in [0, 2^M - 1]$ to the phase signal, $\phi(n)$, provides a phase

offset of $2\pi\frac{p}{2^M}$ radians, assuming the wavetable contains a *single cycle* periodic signal. The offset addition must be performed modulo 2^M as depicted in Figure (4.2.10). The offset DT phase sequence, $\phi'(n)$, is given by $\phi'(n) = \langle \phi(n) + p \rangle_{2^M}$ and the resolution of the M-bit phase offset is $\frac{2\pi}{2^M}$ radians.

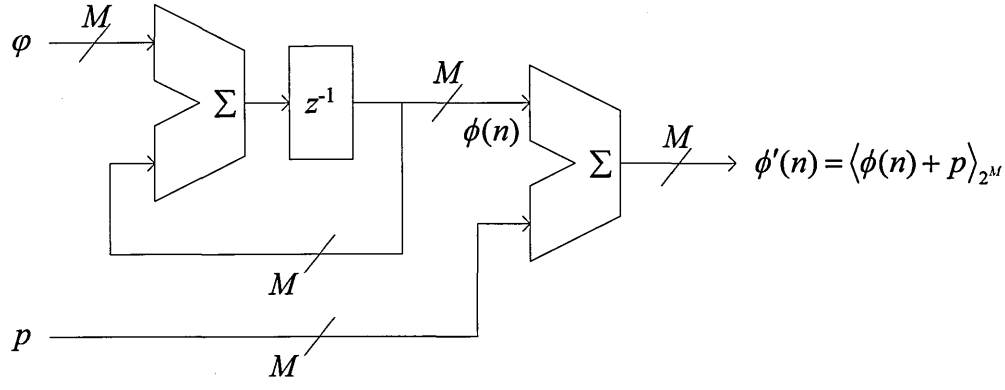


Figure (4.2.10): Phase accumulator with phase offset adder.

Truncating $\phi(n)$ to I-bits reduces the phase resolution of p to $\frac{2\pi}{2^I}$ radians. If p is a time-varying sequence, $\hat{\phi}(n)$, generated by another M-bit phase accumulator with phase increment $\hat{\varphi}$ and sample rate f_s, the resulting sequence is given by:

$$\phi'(n) = \left\langle \phi(n) + \hat{\phi}(n) \right\rangle_{2^M} = \langle n(\varphi + \varphi') \rangle_{2^M} \qquad (4.2.32)$$

The *frequency* of the phase mapped sequence, $\mathbf{T}[\phi'(n)]$, is therefore the sum of the individual phase sequence frequencies and for the specific case when $\hat{\phi}(n) = k\phi(n)$ with k taking on positive integer values, $k \in \{1,2,3,\ldots\}$, we have:

$$\phi'(n) = \langle (k+1)\phi(n) \rangle_{2^M} = \langle n(k+1)\varphi \rangle_{2^M} \qquad (4.2.33)$$

and observe that the frequency of the phase mapped sequence is multiplied by $(k+1)$. We infer from this discussion that the frequency of the k-scaled phase sequence $\hat{\phi}(n)$ is exactly $\frac{k\varphi f_s}{2^M}$.

4.3 Sampling Synthesis

4.3.1 Overview

In our preceding discussions we have investigated wavetables containing precisely periodic signals. We now consider wavetables filled with numerous cycles of a sampled musical instrument sound recorded at a particular pitch – generally reported as *sampling synthesis* in the literature [Roads, 1996]. Since the sampled sound is captured at a known pitch, we model resynthesis as a *relative* pitch shifting process, where broadly two techniques are reported in the literature – *asynchronous* and *synchronous* pitch shifting. However, both of these techniques effectively represent a sample rate conversion of the tabulated signal.

The frequency scaling implicit in pitch shifting a tabulated signal (whether synchronous or asynchronous) causes an inverse scaling of the temporal features of the signal. Upward pitch shifting causes the signal to be *compressed* in time (i.e. reduced in duration) and downward pitch shifting causes the signal to be *stretched* in time (i.e. increased in duration). The key perceptual consequence of this phenomenon is that features in the attack region are compressed or stretched as a function of pitch shift. This is particularly objectionable with sampled vocal sounds where the underlying formant spectra are rescaled, becoming noticeable with only a few semitones of pitch shift [Massie, 1998]. Multisampling as outlined in section (2.2.1), with small pitch shifts applied to each sample, reduces (but does not eliminate) this effect.

4.3.2 Asynchronous Pitch Shifting

Asynchronous pitch shifting changes the clock rate of the wavetable indexing counter to effect a variable sample rate resynthesis of the tabulated signal according to clock frequency. Each channel therefore requires a dedicated digital-to-analogue converter (DAC), sample clock oscillator (whose frequency relative to the acquisition sample rate determines the resynthesis pitch shift) and reconstruction filter whose cut-off frequency tracks the Nyquist frequency. Massie [Massie, 1985] reports the principal advantage of asynchronous pitch shifting as the elimination of pitch shifting "noise artefacts" observed with synchronous techniques. However, asynchronous wavetable indexing is not conducive to multiplexed memory addressing used in multi-voice implementations to reduce cost. Implementation complexity is increased further by the need for multiple analogue reconstruction filters with cut-off frequency that must track the fundamental frequency to remove alias components. Despite these disadvantages, several successful commercial systems were evident in the early 1980s, including the *Fairlight Computer Musical Instrument* series [Roads, 1996], *E-mu Emulator* and *Emulator 2* [Massie, 1985] and underpin the diverse range of commercial sampling synthesis systems seen today.

4.3.3 Synchronous Pitch Shifting

Synchronous pitch shifting operates entirely in the digital (DT) domain and employs sample rate conversion by fractional addressing as outlined in section (4.1.3). The technique uses a fixed sample rate simplifying time-division access to a central sample memory. Post-processing of pitch-shifted signals occurs entirely in the digital domain before conversion into the analogue domain with a *single* DAC and reconstruction filter. However, there are distinct differences with phase increment and phase sequence data partitioning compared to the single cycle wavetable case. The phase increment

fractional component determines the pitch shift *resolution* and the integer part determines the maximum *upward* pitch shift in *octaves*. The phase sequence is partitioned into integer and fraction components as before, with the integer component addressing the wavetable in a "single shot" manner at an effective sample rate controlled by the fractional phase increment according to Eq. (4.1.11), with φ taking on fractional values. The integer address sequence (which spans the whole wavetable address space) may contain "looped" sections which cyclically repeat carefully chosen segments of the wavetable to provide sustain of the resynthesised sound [Massie, 1998]. The fraction component controls interpolation of the wavetable data. In the simplest embodiments, the phase fraction information is discarded (i.e. phase truncation) and no interpolation is applied. A fractional wavetable fetch is then approximated by the tabulated value addressed by the integer component. This is known as *drop sample tuning* in the literature and is equivalent to a zero-order hold interpolation filter in the sample rate conversion model. The *normalised* frequency response of the zero-order hold is a function of U (see section (4.1.5)) as determined by the number of fraction bits in the phase sequence and given by [Massie, 1998; Lyons, 2004]:

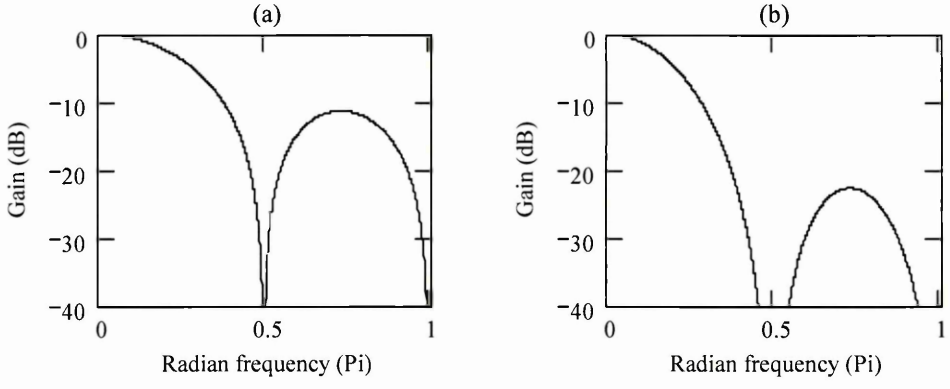
$$|H(\omega)| = \frac{1}{U} \frac{\sin\left(\frac{\omega U}{2}\right)}{\sin\left(\frac{\omega}{2}\right)} \tag{4.3.1}$$

Figure (4.3.1a) depicts the zero-order hold frequency response for $U = 4$ over a frequency range normalised to the up-sampled Nyquist frequency (π), placing the original Nyquist frequency at $\frac{\pi}{4}$. This frequency response exhibits poor rejection of alias images outside the passband with a peak sidelobe attenuation of only –11dB. There are zeros at integer multiples of the original sample frequency ($\frac{\pi}{2}$) and so alias images

of low frequency components in the tabulated signal are well suppressed. Fractional addressing using linear interpolation is equivalent to a first-order hold interpolation filter whose normalised frequency response is given by [Lyons, 2004]:

$$|H(\omega)| = \left(\frac{1}{U} \frac{\sin\left(\frac{\omega U}{2}\right)}{\sin\left(\frac{\omega}{2}\right)} \right)^2 \qquad (4.3.2)$$

and illustrated in Figure (4.3.1b) for $U = 4$. This frequency response exhibits better rejection of alias images outside the passband with a peak sidelobe attenuation of – 23dB. There is better suppression of alias images near multiples of the original sample frequency, giving improved performance when the tabulated signals are oversampled.



Figures (4.3.1a) and (4.3.1b) : (a) – Zero-order and (b) – first-order hold frequency responses for $U = 4$ *over a frequency range normalised to the up-sampled Nyquist frequency (*π*), placing the original Nyquist frequency at* $\frac{\pi}{4}$.

4.3.4 Interpolation Filtering

Higher order interpolation filters give improved alias image rejection at the expense of greater computational complexity. Smith and Gossett [1984] present a time-varying sample rate conversion technique appropriate to sampling WLS. The Smith and Gosset

interpolator computes each output sample from the dot product of a wavetable sample vector and a set of filter coefficients obtained from a lookup table.

The prototype filter coefficients may be pre-computed using a large U value and tabulated in reordered form [Crochiere and Rabiner, 1983]. During operation, the sub-phase of the filter is chosen according to the fractional part of the phase accumulator output sequence on each sample. In general, the filter is described by the linear time-variant filter response function [Proakis and Manolakis, 1996], thus:

$$g(n,m) = h(nI + \langle mD \rangle_U) \tag{4.3.3}$$

where $h(n)$ is the impulse response of the FIR sample rate conversion filter with an *ideal* low-pass frequency response:

$$H(\omega) = \begin{cases} U & \omega \in \left[0, \min\left(\frac{\pi}{D}, \frac{\pi}{U}\right)\right] \\ 0 & \text{otherwise} \end{cases} \tag{4.3.4}$$

If we set the length of the FIR filter to an integer multiple of U, say KU, the set of coefficients $\{g(n,m)\}$ for each $m = 0, 1, 2, \ldots, U-1$, will contain K elements. Since $g(n,m)$ is periodic with period U, the output sample, $y(m)$, is given by:

$$y(m) = \sum_{n=0}^{K-1} g\left(n, m - \left\lfloor \frac{m}{U} \right\rfloor U\right) \mathbf{T}\left[\left\lfloor \frac{mD}{U} \right\rfloor - n\right] \tag{4.3.5}$$

where $\left\lfloor \frac{mD}{U} \right\rfloor$ is the integer part of the phase accumulator output at time index m. Eq. (4.3.5) computes the dot product of K consecutive wavetable values and K filter coefficients $g\left(n, m - \left\lfloor \frac{m}{U} \right\rfloor U\right)$ with $n = 0, 1, 2, \ldots, K-1$. In practice, the set of K coefficients is determined by the fractional component of the phase accumulator output which effects a page address into the coefficient lookup table. Figure (4.3.2) illustrates a process model for this algorithm suggested by Massie [1998].

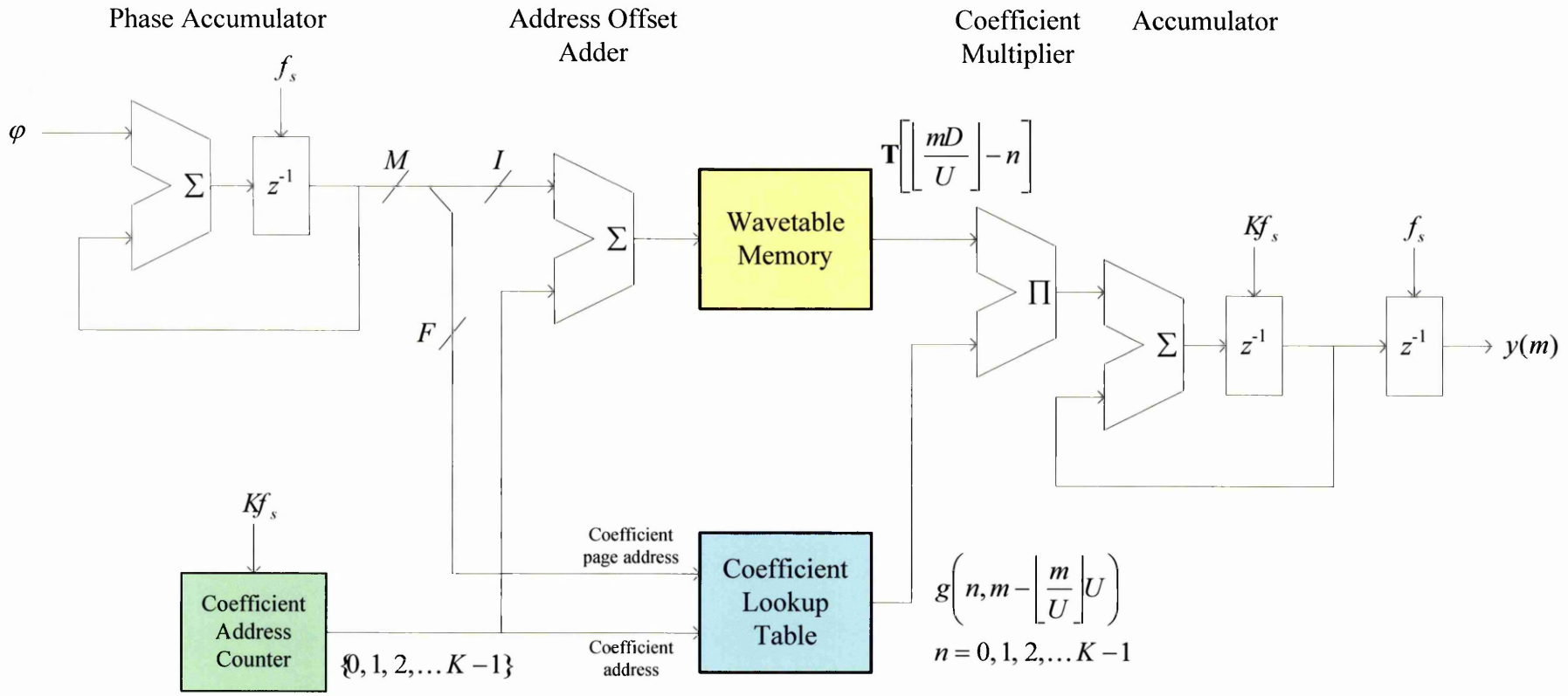


Figure (4.3.2): Sample rate conversion of a tabulated signal using a K sample interpolation filter [Massie, 1998].

4.3.5 Pitch Shift Resolution and Phase Fraction Field Width

We conclude by considering the fractional phase resolution needed in a pitch shift application, as this in conjunction with the number of filter coefficients, K, dictates the size of the coefficient lookup table. As in our previous discussion, we let F denote the number of fraction bits in the phase increment. From section (4.1.3) we know that the fractional phase increment, φ, can be considered as a pitch shift factor with respect to the original pitch of the sampled sound assuming the same sample rate is used throughout the system. (e.g. $\varphi = 2$ causes an *upward* pitch shift of 1 octave and $\varphi = 0.5$ causes a *downward* pitch shift of 1 octave.) If f represents the original frequency of the sampled sound, f' represents the pitch shifted frequency corresponding to a 2^{-F} (i.e. minimum change for F fraction bits) change in φ and Δf represents the corresponding *frequency change*, then:

$$\frac{\Delta f}{f} = \frac{f' - f}{f} = \frac{f'}{f} - 1 \tag{4.3.6}$$

A c cent pitch change with $c \in \Re$ corresponds to a frequency ratio of $2^{\frac{c}{1200}}$ giving:

$$\frac{\Delta f}{f} = 2^{\frac{c}{1200}} - 1 \tag{4.3.7}$$

Observing that $c > 0 \Rightarrow \Delta f > 0$ and $c < 0 \Rightarrow \Delta f < 0$ gives:

$$\frac{f'}{f} = \begin{cases} 2^{\frac{c}{1200}} & c > 0 \\ 2^{-\frac{c}{1200}} & c < 0 \end{cases} \tag{4.3.8}$$

From Eq. (4.1.10) we have $\frac{f'}{f} = \frac{\varphi'}{\varphi}$, where $\varphi' = \varphi + 2^{-F}$ for $\Delta f > 0$ and $\varphi' = \varphi - 2^{-F}$ for $\Delta f < 0$. Substituting for $\frac{f'}{f}$ in Eq. (4.3.8) we obtain the expression:

$$2^{-F} = \begin{cases} \varphi\left(2^{\frac{c}{1200}} - 1\right) & c > 0 \\ \varphi\left(1 - 2^{\frac{c}{1200}}\right) & c < 0 \end{cases} \qquad (4.3.9)$$

which simplifies to a single expression since $\left(2^{\frac{c}{1200}} - 1\right) \approx \left(1 - 2^{\frac{c}{1200}}\right)$ for $|c| \leq 10$:

$$F \approx -\log_2 \varphi - \log_2\left(2^{\frac{c}{1200}} - 1\right) \qquad |c| \leq 10 \qquad (4.3.10)$$

By defining a maximum *downward* pitch shift of $d_{\max}$ octaves we thereby set a minimum φ value given by $\varphi_{\min} = 2^{-d_{\max}}$ and obtain:

$$F \geq d_{\max} - \left\lfloor \log_2\left(2^{\frac{c}{1200}} - 1\right) \right\rfloor \qquad |c| \leq 10 \qquad (4.3.11)$$

where the floor function takes account of F only taking on integer values.

For $c = 1$ and $c = 5$, which represent a 1 and 5 cent pitch tuning error respectively, we have $F \geq d_{\max} + 11$ and $F \geq d_{\max} + 9$. If we require precise beat frequency tuning between two oscillators as discussed in section (4.2.2), the condition on F becomes $F \geq d_{\max} + 20$ for a frequency resolution of ≈ 0.01 Hz at 7040 Hz (A8) corresponding to a ≈ 0.002 cent pitch tuning error. (The frequency resolution improves (i.e. f_r reduces) with reducing frequency.)

4.4 Conclusions

This chapter has introduced the concept of wavetable lookup synthesis (WLS) and presented a taxonomy of wavetable classes differentiated principally by the means used to fill the wavetable. We have reviewed pre-computed wavetable filling techniques and investigated the concept of sampling both single-cycle periodic and multi-cycle aperiodic tabulated signals. This leads to the concept of phase accumulation and the sample rate conversion perspective of pitch shifting a tabulated signal which underpins the *sampling synthesis* subclass. Phase accumulation WLS has been introduced, focussing particularly on phase continuity, frequency resolution and optimal phase-amplitude mapping where we have presented respective theoretical treatments appropriate to computer music requirements.

We conclude that fractional WLS (with implicit phase truncation and hence interpolated wavetable lookup) provides optimal frequency control resolution simultaneous with practicable wavetable lengths. In Chapter 5 we investigate interpolated WLS and present a comparative assessment of interpolation algorithms based on computer simulation. In Chapter 6 we develop the concept of *phase domain* processing introduced in section (4.2.5) as a means of generating harmonic and inharmonic phase sequences which underpin implementation of the HAS and PAS processing models introduced in Chapter 2.

Chapter 5 Interpolated Phase Mapping

5.1 Introduction

In this chapter we investigate interpolated phase-amplitude mapping where the fractional component of a partitioned phase sequence is used to interpolate the wavetable indexing operation according to a particular interpolation algorithm. The concepts of phase truncation, wavetable fractional addressing and interpolation are reviewed and developed. We have reviewed non-truncated (i.e. optimum) phase mapping in section (4.2.4) where *each* phase sequence value within the set of 2^M possible phase states maps to a *unique* location in the wavetable. This condition produces optimal phase-amplitude mapping with no distortion components in the synthesised signal spectrum peculiar to the phase accumulation frequency synthesis process. However, non-truncated phase mapping imposes an impractically large wavetable memory overhead (i.e. 2^M locations) given the phase accumulator word length, M, needed for computer music frequency control resolution.

Truncated phase mapping (TPM) indexes a smaller wavetable using a truncated phase accumulator output word. Compared to the optimal length of 2^M locations, the wavetable size halves with each truncated bit. However, TPM introduces errors into the phase-amplitude mapping process due to some phase-amplitude mappings being approximated to the nearest tabulated value. This causes distortion of the amplitude signal and erroneous spurs in the corresponding spectrum which are extremely frequency (i.e. phase increment) dependent. The fractional phase information (which is discarded with truncated phase mapping) may be used to interpolate (i.e. fractionally address) the wavetable lookup reducing the magnitude of these distortion components at the expense of increased computation.

In this chapter we investigate wavetable phase mapping using a phase sequence partitioned into integer and fraction components. This suggests a fixed-point fractional phase representation thereby enabling interpolation of the tabulated data. We begin by reviewing zero-order interpolation (i.e. phase truncated wavetable lookup with *discarded* phase fraction information) which causes *fractional* phase-amplitude mappings to be approximated to the *lowest* tabulated value. In section (4.2.4) we showed that optimal phase-amplitude mapping is obtained when a wavetable of length 2^M samples is indexed by *all* M phase bits and represents a limiting condition corresponding to zero phase-amplitude mapping error.

We substantiate our investigation with computer simulation of reviewed interpolation phase mapping techniques using performance metrics and simulation conditions which we develop in section (5.4). The simulation results presented in section (5.5) represent the focus of this chapter.

5.1.1 Truncated Phase Mapping

We illustrate the effects of truncated phase mapping (which can be viewed as a *zero-order interpolation* of the tabulated data) by simulating a 12-bit phase accumulator whose output sequence is truncated to 4-bits indexing a 2^4 location wavetable containing a single sinusoid. Simulation results are presented in Figures (5.1.1) through (5.1.3) using parameter values contrived to exaggerate the principal effects. The intentionally poor resolution of the truncated phase sequence exposes phase errors, shown in Figure (5.1.1c), which lead directly to phase-amplitude mapping errors compared to the reference (i.e. optimum) case when all 12 phase bits index a 2^{12} location wavetable.

The corresponding phase mapped reference and phase truncated amplitude signals are shown in Figures (5.1.2a) and (5.1.2b), respectively. The amplitude error sequence

shown in Figure (5.1.2c) clearly confirms the presence of signal distortion in the time domain, with Figure (5.1.3a) illustrating the corresponding amplitude error spectrum. This spectrum exhibits a large number of lines directly attributable to phase-amplitude mapping errors consequential to the phase truncation process. The spectrum corresponding to the reference sinusoid is illustrated in Figure (5.1.3b) for comparison. There are no spurious spectrum lines evident.

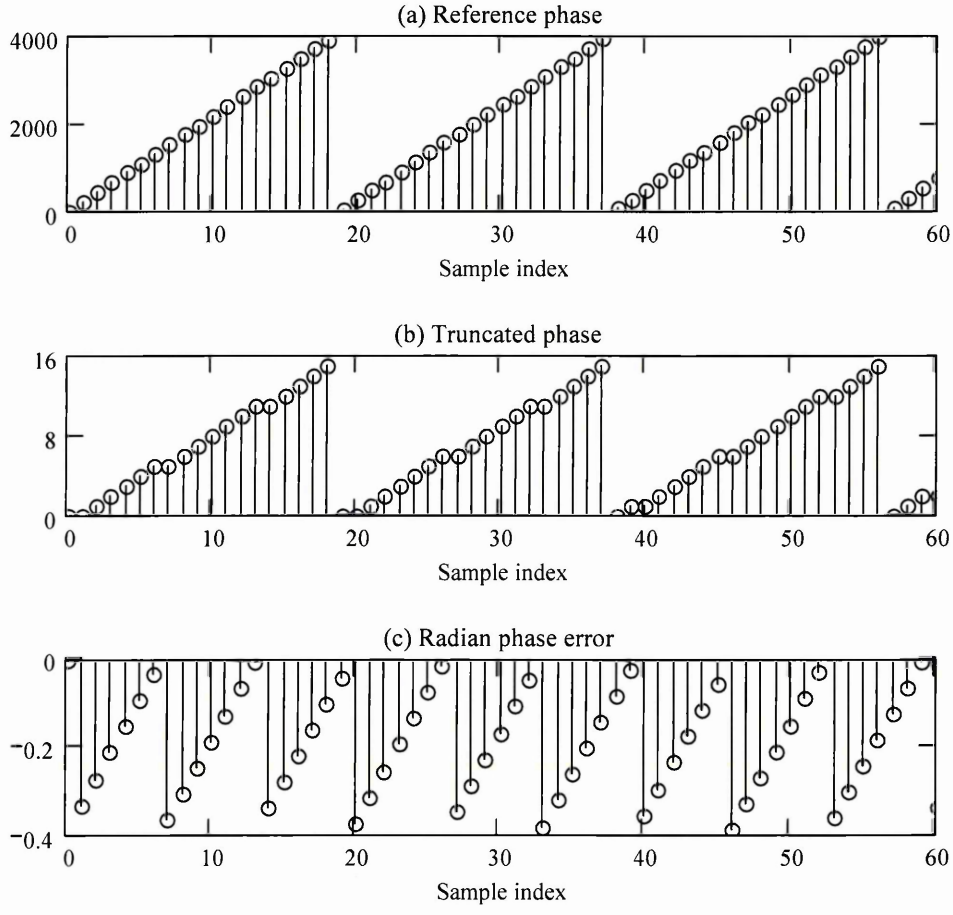


Figure (5.1.1): Example phase sequences. (a) – Reference phase sequence with $M = 12$ *and* $\varphi = 217$. *(b) – 4-bit truncated phase sequence (i.e. 8-bits truncated). (c) – Phase error sequence in radians.*

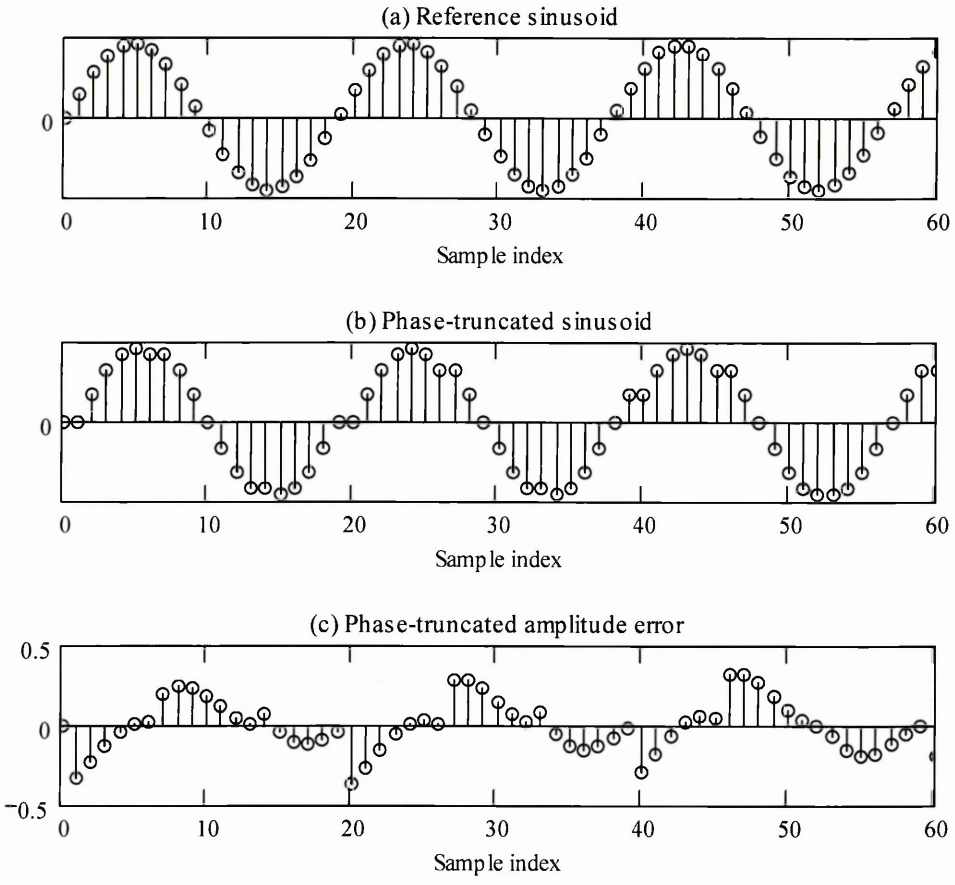


Figure (5.1.2): Phase mapped amplitude sequences corresponding to the phase sequences of Figure (5.1.1). (a) – Reference sinusoid with $L = 2^M$. *(b) – Phase truncated sinusoid with* $L = 2^4$. *(c) – Amplitude error sequence.*

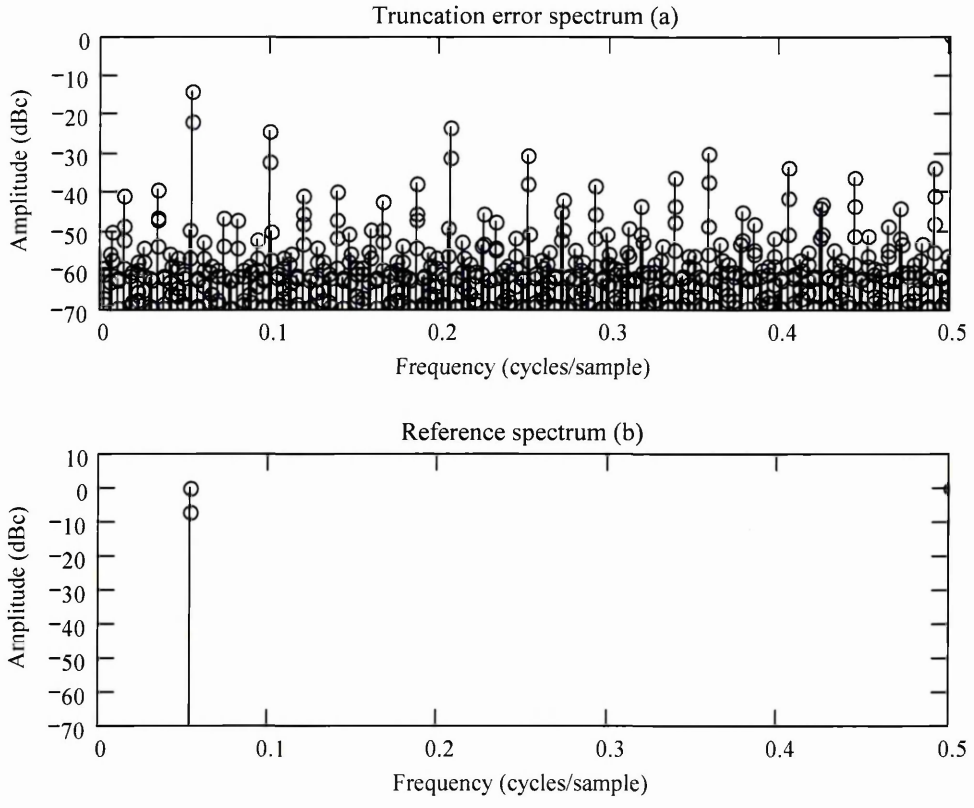


Figure (5.1.3): Phase truncation and reference error spectra corresponding to Figures (5.1.2c) and (5.1.2a). Stem markers ○ indicate the abundance of closely spaced spectrum lines in the phase truncated error spectrum.

5.1.2 Fractional Phase Representation

We now consider truncation of the M-bit phase accumulator output sequence to an I-bit sequence, with $I < M$, which indexes a wavetable of length $L = 2^I$. This yields a partitioned phase sequence comprising an I-bit *integer* field, $\phi_I(n)$, and an F-bit *fraction* field, $\phi_F(n)$, which represents the *normalised* phase fraction $\alpha(n) \in [0, 1)$. Hypothetically, the fraction field may be further truncated from the available $(M - I)$-bits to suit a particular arithmetic processing word size as illustrated in Figure (5.1.4). In general, we have $M = \left(I + \hat{F}\right) = (I + F + R)$, where R denotes the discarded fraction bits from an *available* $\hat{F}$-bit fraction field to give an F-bit *truncated fraction* field.

$R > 0$ is typical when we require frequency resolution consistent with large M (determined by M and f_s) but do not require full phase fraction precision in an interpolated phase mapping computation. For our present discussion we assume $R = 0$ and so $F = \hat{F}$.

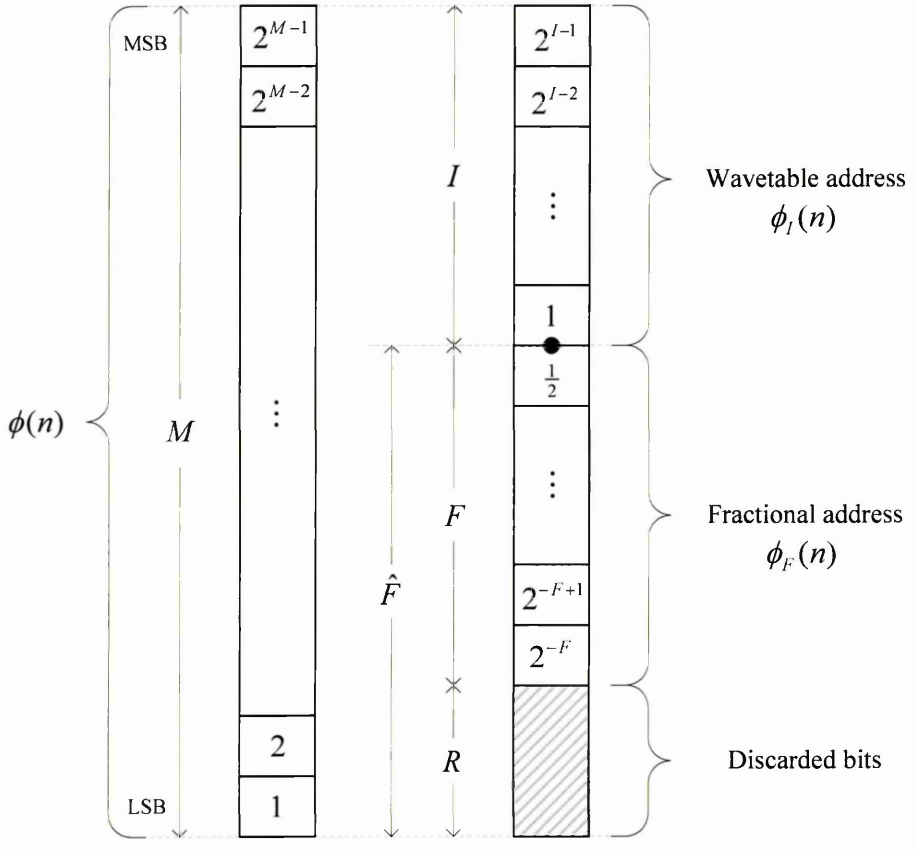


Figure (5.1.4): Phase word partitioning showing truncation of the fraction field.

We now present expressions which model the phase accumulation process and define the non-truncated, truncated and fractional phase sequences, thus:

$$\phi(n) = \langle n\varphi + \phi(0) \rangle_{2^M} \in [0, 2^M - 1] \tag{5.1.1}$$

$$\phi_I(n) = \left\lfloor \frac{\phi(n)}{2^F} \right\rfloor \in [0, 2^I - 1] \tag{5.1.2}$$

$$\phi_{Ir}(n) = \left\langle \left\lfloor \frac{\phi(n)}{2^F} + 0.5 \right\rfloor \right\rangle_{2^I} \in [0, 2^I - 1] \tag{5.1.3}$$

$$\phi_F(n) = \left(\phi(n) - 2^F \phi_I(n)\right) \in [0, 2^F - 1] \tag{5.1.4}$$

$$\alpha(n) = \frac{\phi_F(n)}{2^F} \in [0, 1) \tag{5.1.5}$$

where $\phi_{Ir}(n)$ denotes the truncated phase sequence obtained by *rounding* the fractional value as distinct from $\phi_I(n)$ which is obtained by *discarding* the phase fractional field entirely. Eq. (5.1.3) is expressed in modulo 2^I form since the rounding operation can take the result outside the interval $[0, 2^I - 1]$. For the general truncated fraction field case (i.e. $R > 0$) we modify Eqs. (5.1.4) and (5.1.5), thus:

$$\phi_R(n) = 2^R \left\lfloor \frac{\phi(n) - 2^F \phi_I(n)}{2^R} \right\rfloor \in [0, 2^F - 1] \tag{5.1.6}$$

$$\alpha_R(n) = \frac{\phi_R(n)}{2^F} \in [0, 1) \tag{5.1.7}$$

where $\phi_R(n)$ and $\alpha_R(n)$ denote the truncated forms of $\phi_F(n)$ and $\alpha(n)$, respectively. For the specific case, when $R = 0$ we have:

$$\phi(n) \equiv 2^F \phi_I(n) + \phi_F(n) \tag{5.1.8}$$

For $I < M$, this phase sequence is interpreted as the fractional quantity $\phi_I(n) + \alpha(n)$ which *fractionally addresses* the wavetable through interpolation.

A distinction is evident between *truncated phase mapping* (TPM) (i.e. zero-order interpolation) which effects phase-amplitude mapping with a discarded phase fraction field and *interpolated phase mapping* (IPM) which uses the fractional phase information

to interpolate the wavetable lookup. Rounded phase mapping (RPM), which can be viewed as an alternative form of zero-order interpolation, provides a smaller phase quantisation error on TPM since some of the phase fraction information is used to round the truncated phase value.

5.2 Fractional Wavetable Addressing and Polynomial Interpolation

5.2.1 Preliminaries

Implementation of phase-amplitude mapping by means of a fractionally addressed wavetable with unit-spaced tabulation, requires that we interpolate values of the underlying function *between* tabulated points according to a fractional phase representation as discussed in section (5.1.2) and depicted in Figure (5.1.4). In essence, interpolating non-tabulated (i.e. fractionally addressed) values requires construction of an *interpolating polynomial* that passes through each sample within a localised subset of tabulated samples in the vicinity of the sample being interpolated. The interpolating polynomial therefore *locally* approximates the underlying continuous-time function tabulated in the wavetable at discrete phase points and may be expressed in several mathematically equivalent forms which include power series, Lagrange and Newton representations. Appendix A reviews the analytical basis of polynomial interpolation applied to the interpolation of a well-behaved function, $f(x)$, tabulated at distinct values of x.

Referring to the interpolation analysis presented in Appendix A, we define the generalised fractional address, x, in terms of the phase sequence, $\phi(n)$, partitioned into an I-bit integer field, $\phi_I(n) = \left\lfloor \frac{\phi(n)}{2^F} \right\rfloor$, and an F-bit fraction field, $\alpha(n) = \frac{\phi(n)}{2^F} - \left\lfloor \frac{\phi(n)}{2^F} \right\rfloor$, hence modifying Eq. (5.1.8) thus:

$$x = \frac{\phi(n)}{2^F} = \phi_I(n) + \alpha(n) \qquad (5.2.1)$$

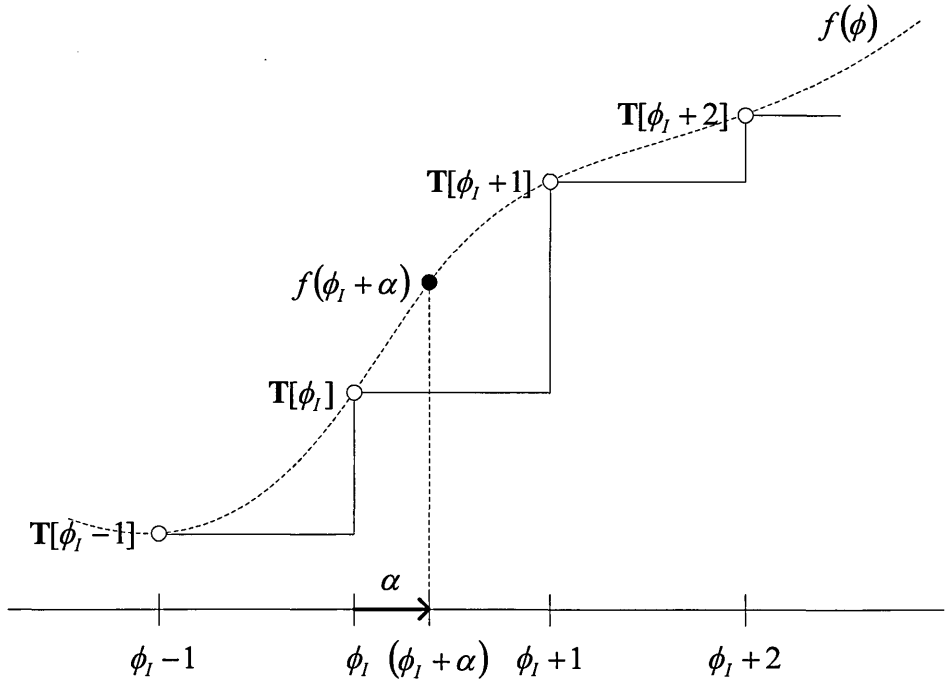
where the *integer* and *fraction* components of $\phi(n)$ are represented by $\phi_I(n) \in [0, 2^I - 1]$ and $\alpha(n) \in [0, 1)$, respectively. In the subsequent discussion we drop sequence time-indices for brevity and use ϕ_I and α to denote the sequences $\phi_I(n)$ and $\alpha(n)$, respectively.

5.2.2 The Cubic Interpolation Polynomial

By way of an example, we consider the cubic Lagrange interpolation polynomial (i.e. $N = 3$) interpolating x with the tabulated data set $\{f(x_0), f(x_1), f(x_2), f(x_3)\}$ for $x \in [x_0, x_1]$ using the notation of Appendix A. In the WLS context, this corresponds to four (i.e. $N+1$) wavetable read operations which interpolates $x = (\phi_I + \alpha)$ with the wavetable sample set $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1], \mathbf{T}[\phi_I + 2], \mathbf{T}[\phi_I + 3]\}$). (We further relax denotation rigour by noting that strictly the wavetable is indexed modulo 2^I and so $\mathbf{T}[\phi_I \pm a]$ is properly denoted as $\mathbf{T}[\langle \phi_I \pm a \rangle_{2^I}]$ where a is an arbitrary integer offset.) We note from the interpolation error discussion presented in Appendix A that $x = (\phi_I + \alpha)$ is placed in the *first* sub-interval of the sample set leading to a *non-optimal* error magnitude bound.

The fractional phase value, α, spans the sub-interval $[\phi_I, \phi_I + 1)$ and so for unit-spaced tabulation (i.e. $x = 0, 1, 2, \ldots$) $x_j = \phi_I + j$ in the numerator of Eq. (A-5). From Eq. (5.2.1) we have $x = (\phi_I + \alpha)$ and so the numerator terms in Eq. (A-5) reduce to $(\alpha - j)$. Adding an offset to the wavetable index so that $\phi_I \to (\phi_I - 1)$ interpolates $x = (\phi_I + \alpha)$ with the tabulated sample set $\{\mathbf{T}[\phi_I - 1], \mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1], \mathbf{T}[\phi_I + 2]\}$ as illustrated in

Figure (5.2.1) and places the fractional address in the middle sub-interval of this set, $[\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1]]$, as required for a minimum interpolation error bound.



$$\phi_I \equiv \phi_I(n) = \left\lfloor \frac{\phi(n)}{2^F} \right\rfloor \qquad \alpha \equiv \alpha(n) = \frac{\phi_F(n)}{2^F} = \left(\frac{\phi(n)}{2^F} - \left\lfloor \frac{\phi(n)}{2^F} \right\rfloor \right)$$

Figure (5.2.1): Illustration of the optimal fractional address interval for the cubic interpolation polynomial with $x = (\phi_I + \alpha)$.

5.2.3 The Optimal Order-*N* Polynomial Interpolator

In general, optimal positioning of the interpolation sample set with respect to the fractional address interval requires that we offset each numerator and wavetable index term in Eq. (A-5) by $-\left\lfloor \frac{N-1}{2} \right\rfloor$ which places $x = (\phi_I + \alpha)$ in the *middle* sub-interval of the sample set for all odd N and in the *lower* sub-interval from the middle sample for

all even N. Observing modulo 2^I wavetable indexing, the generalised wavetable read operation now becomes $\mathbf{T}\left[\left\langle \phi_I - \left\lfloor \frac{N-1}{2} \right\rfloor + k \right\rangle_{2^I}\right], \quad k \in [0, N]$.

For unit-spaced sample sets corresponding with wavetable tabulation, the $(x_k - x_j)$ denominator terms in Eq. (A-5) reduce to constant integer values (i.e. $(x_k - x_j) = k - j$), hence for an order-N interpolated output sample, $P_N(\phi_I + \alpha)$, we have:

$$P_N(\phi_I + \alpha) = \sum_{k=0}^{N} \left[\mathbf{T}\left[\left\langle \phi_I - \left\lfloor \frac{N-1}{2} \right\rfloor + k \right\rangle_{2^I}\right] \prod_{\substack{j=0 \\ j \neq k}}^{N} \left(\frac{\alpha + \left\lfloor \frac{N-1}{2} \right\rfloor - j}{k - j} \right) \right] \tag{5.2.2}$$

which reduces to *linear interpolation* as given by Eq. (3.3.4) when $N = 1$ and truncation when $N = 0$. We simplify Eq. (5.2.2) by defining a single scaling factor, $\beta_k(\alpha)$, for each summation term, thus:

$$\begin{aligned} \beta_k(\alpha) &= \prod_{\substack{j=0 \\ j \neq k}}^{N} \left(\frac{\alpha + \left\lfloor \frac{N-1}{2} \right\rfloor - j}{k - j} \right) \\ &= \prod_{\substack{j=0 \\ j \neq k}}^{N} \left(\left(\alpha + \left\lfloor \frac{N-1}{2} \right\rfloor - j \right)(k - j)^{-1} \right), \quad k \in [0, N] \end{aligned} \tag{5.2.3}$$

and so, in general we have:

$$P_N(\phi_I + \alpha) = \sum_{k=0}^{N} \left[\beta_k(\alpha) \mathbf{T}\left[\left\langle \phi_I - \left\lfloor \frac{N-1}{2} \right\rfloor + k \right\rangle_{2^I}\right] \right] \tag{5.2.4}$$

For the cubic interpolation example we have:

$$P_3(\phi_I + \alpha) = \beta_0(\alpha)\mathbf{T}[\phi_I - 1] + \beta_1(\alpha)\mathbf{T}[\phi_I] + \beta_2(\alpha)\mathbf{T}[\phi_I + 1] + \beta_3(\alpha)\mathbf{T}[\phi_I + 2] \tag{5.2.5}$$

where $\beta_0(\alpha) = -\frac{1}{6}\alpha(\alpha-1)(\alpha-2)$, $\beta_1(\alpha) = \frac{1}{2}(\alpha+1)(\alpha-1)(\alpha-2)$,

$\beta_2(\alpha) = -\frac{1}{2}\alpha(\alpha+1)(\alpha-2)$ and $\beta_3(\alpha) = \frac{1}{6}\alpha(\alpha+1)(\alpha-1)$.

Appendix A reviews the Newton interpolation form defined by Eq. (A-11) which expresses an order-N interpolating polynomial in terms of an order $N-1$ polynomial and leads to computational advantages under certain conditions. Following similar reasoning applied in section (5.2.2), we substitute $x = (\phi_I + \alpha)$ in Eq. (A-11) and optimise location of the fractional address interval thereby obtaining the recursive equation:

$$P_N(\phi_I + \alpha) = P_{N-1}(\phi_I + \alpha) + a_N \prod_{j=0}^{N-1}\left(\alpha + \left\lfloor \frac{N-1}{2} \right\rfloor - j\right) \qquad (5.2.6)$$

where the m^{th} divided difference term, a_m, is defined by:

$$a_m = \sum_{k=0}^{m}\left(\mathbf{T}\left[\left\langle \phi_I - \left\lfloor \frac{N-1}{2} \right\rfloor + k \right\rangle_{2^I}\right]\prod_{\substack{j=0 \\ j\neq k}}^{m}(k-j)^{-1}\right), \quad m \in [0, N] \qquad (5.2.7)$$

Eqs. (5.2.6) and (5.2.7) define a recursive algorithm for computing the order-N Newton interpolation in N iterations, requiring $(N+1)$ wavetable read operations. Horner's algorithm (as presented in Appendix A-3) allows the Newton interpolation polynomial form to be expressed in a *reduced-multiplier* form which we exemplify with the cubic interpolation polynomial, thus:

$$P_3(\phi_I + \alpha) = a_0 + [a_1 + [a_2 + a_3(\alpha-1)]\alpha](\alpha+1) \qquad (5.2.8)$$

where the *divided differences*, a_0 to a_3, are given by $a_0 = \mathbf{T}[\phi_I - 1]$,

$a_1 = \mathbf{T}[\phi_I] - \mathbf{T}[\phi_I - 1]$, $a_2 = \frac{1}{2}\mathbf{T}[\phi_I + 1] - \mathbf{T}[\phi_I] + \frac{1}{2}\mathbf{T}[\phi_I - 1]$ and

$a_3 = \frac{1}{6}\mathbf{T}[\phi_I + 2] - \frac{1}{2}\mathbf{T}[\phi_I + 1] + \frac{1}{2}\mathbf{T}[\phi_I] - \frac{1}{6}\mathbf{T}[\phi_I - 1]$ with lookup table indexing offset to ensure optimal placement of the fractional address interval and hence minimum

interpolation error bound. It is evident that each divided difference term is a linear combination of the $(N+1)$ sample set elements with *constant* weightings.

5.2.4 Interpolation Arithmetic Overhead

In the assessment of arithmetic overhead, we are primarily concerned with the number of multiplication and memory read operations. Multiplication count is significant since multiplier architectures impose a much greater area penalty and execution time in VLSI implementations compared to adder architectures, which are generally smaller and faster. Memory read-access and cycle times discourage multiple reads from a single memory in high speed applications consistent with a time-division multiplexed WLS oscillator generating multiple voices. However, the falling cost of high speed memory encourages the use of relatively small lookup tables to eliminate real-time computation. The order-N Lagrange interpolation polynomial given by Eq. (5.2.4) requires:

- $\sum_{0}^{N}(N-1)$ multiplication operations associated with the α terms;
- $(N+1)$ multiplication operations associated with the $(k-j)^{-1}$ terms (each of which is *constant* for a given interpolation order);
- $(N+1)$ multiplication operations associated with the $\beta_k(\alpha)\mathbf{T}$ terms;
- $< N(N+1)$ addition operations associated with the α terms;
- N addition operations associated with the $\beta_k(\alpha)\mathbf{T}$ terms;
- N addition operations associated with the ϕ_l offset terms;
- $N+1$ memory read operations.

Hence, the total multiplication and addition counts are (N^2+2N+1) and $N(N+3)$, respectively and are both of $O(N^2)$. For the cubic interpolation polynomial example, we require sixteen multiplication, fifteen addition and four wavetable read operations.

Algorithmically, Eq. (5.2.4) represents a *linear combination* of $(N+1)$ wavetable samples whose weighting coefficients, $\beta_k(\alpha), k \in [0, N]$, are a function of fractional address, α, and interpolation order, N (which is constant in a given implementation). Computing the $\beta_k(\alpha)$ terms from scratch is computationally intensive requiring N multiplications per term. Alternatively, we may obtain the $\beta_k(\alpha)$ terms directly by indexing $(N+1)$ lookup tables with the phase fraction field, thereby eliminating N multiplication operations at the expense of lookup table memory. In general, for lookup table computed $\beta_k(\alpha)$ terms, we observe a minimum of $(N+1)$ multiplications, N additions, $(N+1)$ wavetable read operations and $(N+1)$ $\beta_k(\alpha)$ table lookup operations to compute the Lagrange interpolation algorithm.

Inspecting Eqs. (5.2.6) and (A-14) we deduce that for $N \geq 2$ the order N reduced-multiplier Newton interpolation polynomial requires:

- $\sum_{i=0}^{N}(N-i)+N-3$ multiplication operations associated with the a_m divided difference terms;
- N multiplication operations for the α weighting terms;
- $\sum_{k=1}^{N} k = (N+1)\frac{N}{2}$ addition operations associated with the a_m divided difference terms;
- N addition operations associated with the P_N summation;
- $(N-1)$ addition operations associated with the α terms;
- N addition operations associated with the ϕ_I offset terms;
- $(N+1)$ memory read operations.

Hence, the total multiplication and addition counts are $\sum_{i=0}^{N}(N-i)+2N-3$ and $(N+1)\frac{N}{2}+3N-1$, respectively. For our example cubic interpolation polynomial, we have nine multiplication operations against sixteen for the Lagrange form. Furthermore, four of these multiplications are by *exactly* $\frac{1}{2}$ and may therefore be effected by arithmetic right shifts, reducing the multiplication count to five.

If the a_m terms in Eq. (5.2.8) are obtained from four distinct wavetables each tabulating a particular a_m term with all tables indexed by ϕ_I, the multiplication count reduces to three. For the order-N, reduced-multiplier Newton interpolation polynomial with coefficient terms computed by table lookup, we observe a minimum of N multiplications and $(N+1)$ distinct wavetables each tabulating a unique divided difference of the underlying wavetable being interpolated. However, this imposes an $(N+1)$ fold increase in wavetable memory length compared to the recursive algorithm of Eq. (5.2.6) which follows a sequential processing model with $(N+1)$ memory read operations from a *single* wavetable memory.

Figures (5.2.2) and (5.2.3) illustrate the respective variation in multiplication and addition operation count with N for three interpolation classes: Newton, Lagrange and Lagrange with coefficients computed by table lookup. We see that the latter form requires the lowest number of multiplication operations for a given N.

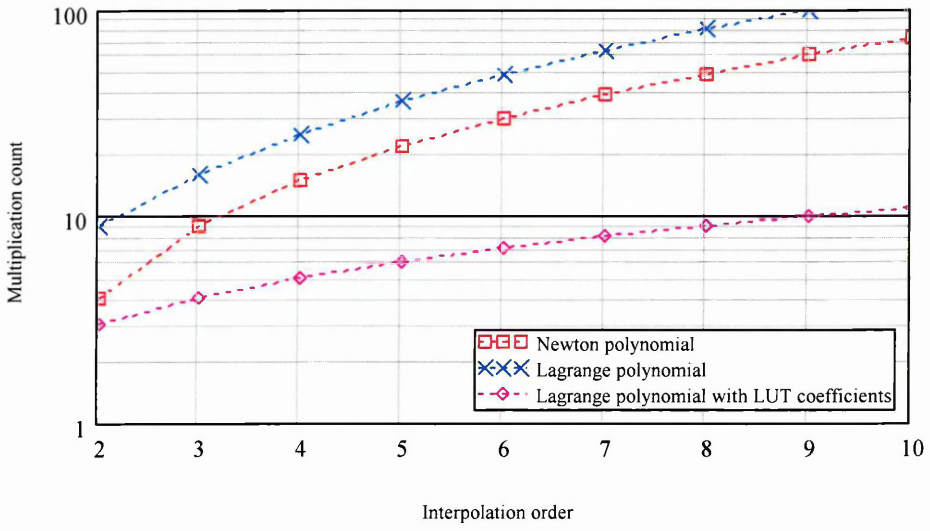


Figure (5.2.2): Multiplication count as a function of interpolation order for three interpolating polynomials.

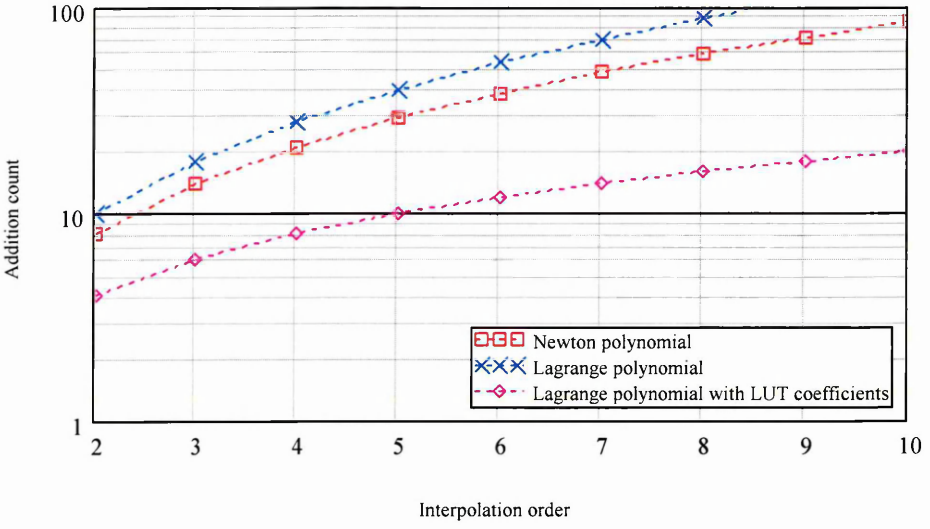


Figure (5.2.3): Addition count as a function of interpolation order for three interpolating polynomials.

5.3 Trigonometric Identity Phase Mapping (TIPM)

This section develops the exploitation of trigonometric identities to reduce memory requirement in the computation of sinusoidal phase-amplitude mapping. This work has been published by the author in [Symons, 2002] as part of this research and subsequently extended here to consider optimal phase word partitioning, memory saving compared to brute force phase mapping and arithmetic precision requirements.

5.3.1 The Trigonometric Addition Identity and Sinusoidal Phase Mapping

The phase sequences, $\phi_I(n)$ and $\phi_F(n)$, combine according to Eq. (5.1.8) to form the non-truncated sequence, $\phi(n)$, suggesting consideration of the trigonometric identities:

$$\sin(A+B) \equiv \sin(A)\cos(B) + \cos(A)\sin(B) \tag{5.3.1}$$

$$\cos(A+B) \equiv \cos(A)\cos(B) - \sin(A)\sin(B) \tag{5.3.2}$$

to effect the phase mapping function. This phase mapping is absolutely exact since the identities are themselves exact (by definition) and all phase information is used. Using Eqs. (5.1.2) and (5.1.4) we decompose the non-truncated phase sequence, $\phi(n)$, expressed in radian form, thus:

$$\begin{aligned}\frac{2\pi}{2^M}\phi(n) &= \frac{2\pi}{2^M}\left(2^F\left\lfloor\frac{\phi(n)}{2^F}\right\rfloor\right)+\frac{2\pi}{2^M}\left(\phi(n)-2^F\left\lfloor\frac{\phi(n)}{2^F}\right\rfloor\right)\\ &= \frac{2\pi}{2^I}\phi_I(n)+\frac{2\pi}{2^M}\phi_F(n)\end{aligned} \tag{5.3.3}$$

Equation (5.3.3) enables an optimal *sinusoidal* phase mapping definition by substituting $(A+B)=\frac{2\pi}{2^M}\phi(n)$, $A=\frac{2\pi}{2^I}\phi_I(n)$ and $B=\frac{2\pi}{2^M}\phi_F(n)$ into Eqs. (5.3.1) and (5.3.2), thus:

$$\sin\left(\frac{2\pi}{2^M}\phi(n)\right) = \sin\left(\frac{2\pi}{2^I}\phi_I(n)\right)\cos\left(\frac{2\pi}{2^M}\phi_F(n)\right)+\cos\left(\frac{2\pi}{2^I}\phi_I(n)\right)\sin\left(\frac{2\pi}{2^M}\phi_F(n)\right) \tag{5.3.4}$$

$$\cos\left(\frac{2\pi}{2^M}\phi(n)\right) = \cos\left(\frac{2\pi}{2^I}\phi_I(n)\right)\cos\left(\frac{2\pi}{2^M}\phi_F(n)\right) - \sin\left(\frac{2\pi}{2^I}\phi_I(n)\right)\sin\left(\frac{2\pi}{2^M}\phi_F(n)\right)$$

(5.3.5)

Figure (5.3.1) illustrates the arithmetic process model associated with Eqs. (5.3.4) and (5.3.5) where four lookup tables indexed by $\phi_I(n)$ and $\phi_F(n)$ provide the sine and cosine terms. Additional arithmetic overhead includes four multiplications, one addition and one subtraction to compute the quadrature sinusoid samples. Figure (5.3.2) illustrates the simplified process model particular to non-quadrature phase mapping, removing two multiplications and one subtraction operation.

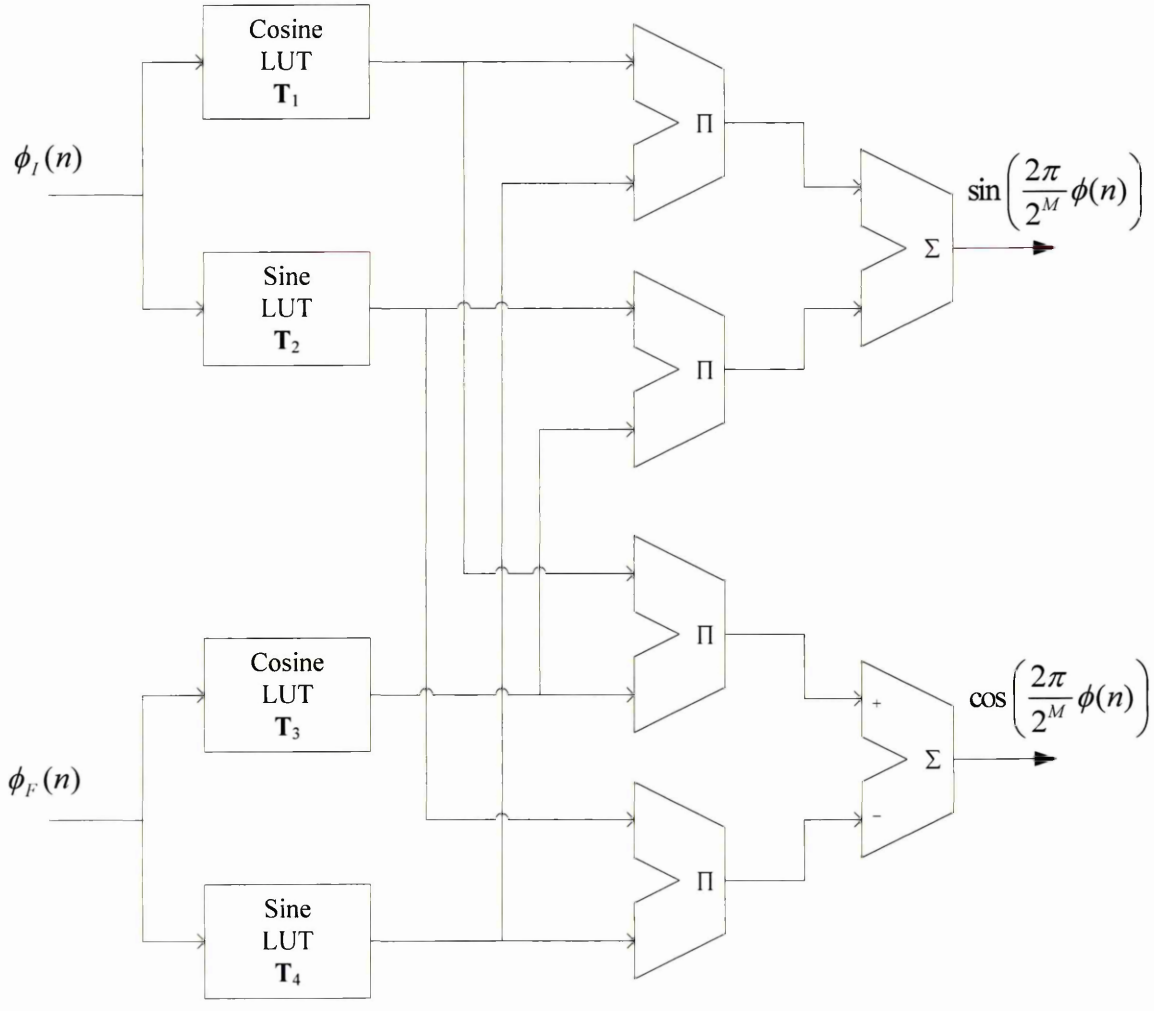


Figure (5.3.1): Arithmetic process model for quadrature trigonometric phase mapping given by Eqs. (5.3.4) and (5.3.5).

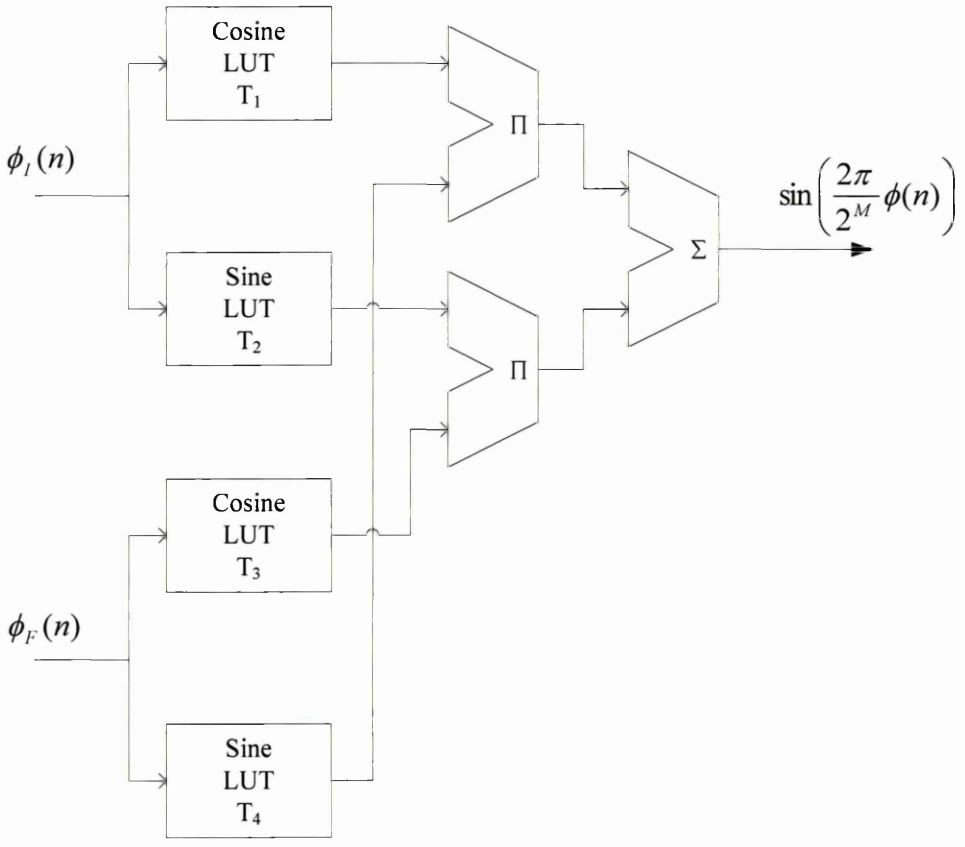


Figure (5.3.2): Arithmetic process model for non-quadrature (single sinusoid) trigonometric phase mapping given by Eq. (5.3.4).

The A and B wavetables are of length 2^I and 2^F samples respectively, with total memory requirement now $\left(2^{I+1}+2^{F+1}\right)$ representing a significant reduction on the brute force value of 2^M samples. We define the four wavetable functions, thus:

$$\left.\begin{aligned}\mathbf{T}_1[a] &= \cos\left(\frac{2\pi}{2^I}a\right)\\ \mathbf{T}_2[a] &= \sin\left(\frac{2\pi}{2^I}a\right)\end{aligned}\right\} \quad a \in [0, 2^I - 1] \qquad (5.3.6)$$

$$\left.\begin{aligned}\mathbf{T}_3[a] &= \cos\left(\frac{2\pi}{2^M}a\right)\\ \mathbf{T}_4[a] &= \sin\left(\frac{2\pi}{2^M}a\right)\end{aligned}\right\} \quad a \in [0, 2^F - 1] \qquad (5.3.7)$$

observing that lookup tables $\mathbf{T}_3$ and $\mathbf{T}_4$ contain a sinusoid *sub-cycle* (i.e. 2^{-I} of one cycle). Since the angle summation formulae represented by Eqs. (5.3.1) and (5.3.2) are mathematical identities (i.e. absolutely exact), we hypothesise that trigonometric identity phase mapping (TIPM) provides optimum signal-to-noise-ratio (SNR[1]) constrained only by sample quantisation and computation round-off errors.

5.3.2 Optimal Phase Word Partitioning

We now consider optimal partitioning of the phase word (i.e. the relative magnitude of I and F) so as to minimise the total wavetable memory requirement compared to the brute force value of 2^M samples. We first define the "memory reduction" ratio, η, as the ratio of total wavetable memory size using trigonometric phase mapping to the corresponding brute force value, thus:

$$\eta = \frac{2(2^I + 2^{M-I})}{2^M} \tag{5.3.8}$$

Setting $\frac{d\eta}{dI} = \frac{2\ln(2)}{2^M}\left(2^I - 2^{M-I}\right) = 0$ gives the minimum of this function as $I = F = \frac{M}{2}$.

Figure (5.3.3) illustrates the behaviour of η according to Eq. (5.3.8) as I varies over the interval $[1, M-1]$. Optimum reduction ratio is evident when $I = F = \frac{M}{2}$.

[1] See section (5.4) for a definition of this performance metric in the context of phase-amplitude mapping error.

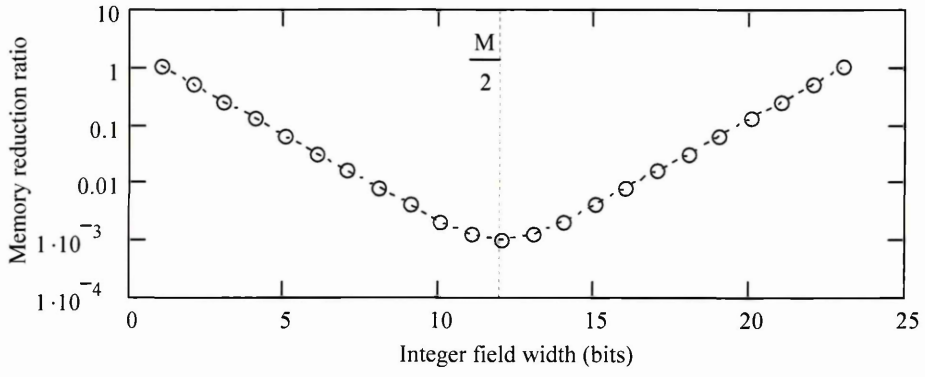


Figure (5.3.3): Wavetable memory reduction ratio as a function of integer field width.

We deduce that maximum wavetable memory saving occurs when the phase word is partitioned equally when all four lookup tables have length $\frac{M}{2}$ samples. The memory reduction ratio is then $2^{\left(2-\frac{M}{2}\right)}$ and improves as M increases.

5.3.3 The Reduced-Multiplier Quadrature TIPM Form

Eqs. (5.3.4) and (5.3.5) can be manipulated as a *pair* to give an equivalent but reduced multiplier form applicable to quadrature sinusoid generation. Arithmetic saving comes from identifying the common term $\cos(A)[\cos(B)+\sin(B)]$, thus:

$$\sin(A+B) = \cos(A)[\cos(B)+\sin(B)] + \cos(B)[\sin(A)-\cos(A)] \qquad (5.3.9)$$

$$\cos(A+B) = \cos(A)[\cos(B)+\sin(B)] - \sin(B)[\sin(A)+\cos(A)] \qquad (5.3.10)$$

requiring four table lookup operations, three multiplications, three additions and two subtractions. Exchanging a multiply for one addition and two subtraction operations is advantageous in VLSI implementations due to silicon area saving. Figure (5.3.4) illustrates the arithmetic process model for quadrature phase mapping using Eqs. (5.3.9) and (5.3.10).

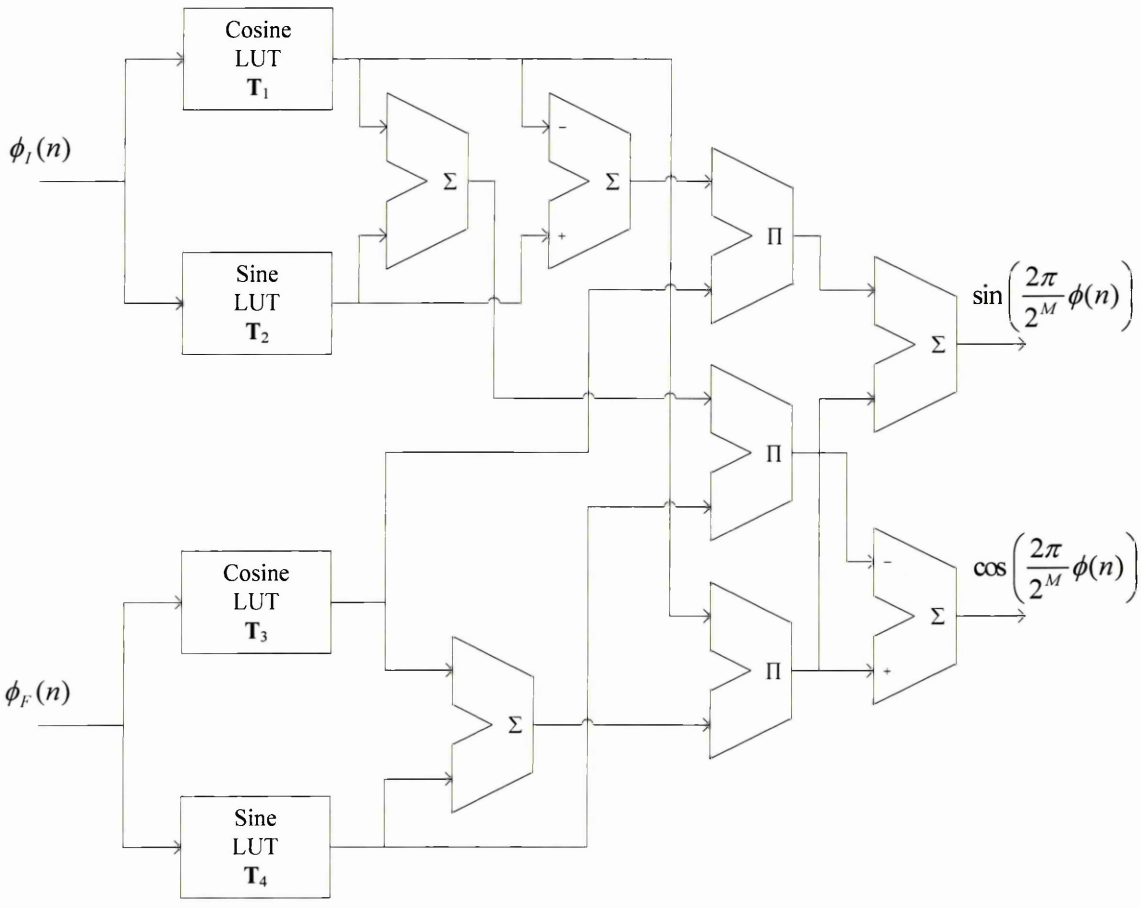


Figure (5.3.4): Arithmetic process model of the reduced-multiplier TIPM algorithm which is applicable to precision quadrature sinusoid synthesis only.

5.3.4 Arithmetic Precision and Sample Word Size

For the condition $I = F$, it is clear from Eqs. (5.3.7) that the fractional phase ranges over smaller positive intervals as M increases. The corresponding sine and cosine amplitude values are therefore always positive and tend to zero and one respectively as M increases. Hence, the fractional sine and cosine terms obtained from lookup tables $\mathbf{T}_4$ and $\mathbf{T}_3$ require fewer bits for accurate representation within a fixed-point number system. It is clear from the Taylor series expansions of $\sin(x)$ and $\cos(x)$ that $\sin(x) \to x$ and $\cos(x) \to 1$ as $x \to 0$. The *maximum* fractional phase angle is $2\pi\dfrac{(2^F - 1)}{2^M} \cong 2\pi 2^{-I}$ radians for $2^F >> 1$ and so the corresponding fractional sine and cosine amplitude values are $\sin(2\pi 2^{-I}) \approx 0$ and $\cos(2\pi 2^{-I}) \approx 1$, respectively.

A 2's complement fixed-point number representation of b-bits has a resolution (quantisation interval) of $2^{-(b-1)}$ normalised to a full scale range of $[-1, (1-2^{-(b-1)})]$. Hence, an arbitrary value, a, lying within the *positive* half of the range (i.e. $a \in [0, (1-2^{-(b-1)})]$), requires $b' = \log_2\left\lceil \dfrac{a}{2^{-(b-1)}} \right\rceil$ bits of the available $(b-1)$-bit fraction field to represent it, with the remaining $b-1-b'$ bits set to 0. Similarly, an arbitrary value, a, lying very close to positive full-scale (i.e. $a \to (1-2^{-(b-1)})$), requires $b' = \log_2\left\lceil \dfrac{1-a}{2^{-(b-1)}} \right\rceil$ bits of the available $(b-1)$-bit field to represent it, with the remaining $b-1-b'$ bits set to 1. We therefore observe a reduction in the number of bits required to accurately represent our fractional sine and cosine amplitude values.

We can assess the saving in arithmetic word size by considering an example where $M = 24$ as concluded in section (4.2.2) and hence $I = 12$ for optimum lookup table memory utilisation as concluded in section (5.3.2), whereupon $\sin(2\pi 2^{-I}) \approx 0.001534$

and $\cos(2\pi 2^{-I}) \approx 0.999999$. The corresponding 24-bit 2's complement fixed-point binary representations are therefore (**0**) 000 0000 0011 0010 0100 0100 and (**0**) 111 1111 1111 1111 1111 0110, respectively where (**0**) denotes the sign bit value and the shaded areas illustrate the significant bits. Hence, we have $b' = 14$ and $b' = 4$ for the fractional sine and cosine amplitude values, respectively. This saving in arithmetic word size affords a significant reduction in multiplier gate count in VLSI implementations. For our above example, we now require 24 by 14 and 24 by 4 bit multipliers for the fractional sine and cosine terms, respectively. Finally, for arithmetic processing where $b < 20$, $M = 24$ and $I = F = 12$, the fractional cosine amplitude term cannot be represented within the available resolution and so becomes superfluous, thereby removing one lookup table and multiplication operation.

5.4 Simulation Development

In this section we consider a simulation framework that supports a qualitative assessment and comparison of interpolated WLS phase mapping algorithms using numerical computer models implemented in Mathcad version 11.2a and presented in Appendix B. We are also concerned with the definition of critical parameter values (e.g. phase increment, the number of samples simulated and tabulated signal spectral characteristics) which ensure an accurate and consistent simulation of *worst case* error performance. This section develops the qualitative evaluation work of Moore [1977] in the simulation of rudimentary sinusoidal WLS and Dannenberg [1998] in the simulation of multi-harmonic (i.e. spectrally rich) WLS. We generalise and develop this work to provide a "modelling toolbox" that enables simulation and assessment of particular WLS configurations and tabulated signal spectra with independent variation of all control parameters.

The phase sequences defined by Eqs. (5.1.1) through (5.1.5) underpin all the Mathcad models and together with the particular interpolation algorithm under evaluation (e.g. linear interpolation), compute a simulated output vector which is compared to a reference vector computed to full arithmetic precision. The simulation is run for a specific number of samples designed to ensure the output vector contains a near-integral number of cycles. Simulation results are presented and discussed in section (5.5) for several interpolating phase mapping algorithms (including the trigonometric identity algorithm) using a range of test conditions typical of real-world situations.

We use two qualitative metrics – the signal-to-noise ratio (SNR) computed over N_s samples, $\mathrm{SNR}(N_s)$, as defined in section (5.4.1) and the spectrum of the amplitude error signal defined as the difference between the interpolated signal and a reference

signal having the same frequency, phase and spectral composition computed to full arithmetic precision. In this context we define "noise" as amplitude error components in the simulated signal corresponding to both phase truncation error and amplitude quantisation effects. The magnitude of phase truncation noise is reduced by interpolative phase mapping but, in general, can only be minimised. Quantisation noise sources take on two distinct forms – noise due to intended simulation of amplitude quantisation (e.g. 16-bit fixed-point linear quantisation) and "computation noise" due to the finite resolution of the full precision arithmetic used in the Mathcad modelling environment. We employ full precision computation when we wish to determine SNR performance "uncontaminated" by amplitude quantisation noise (i.e. determining SNR due to phase truncation errors *alone*).

5.4.1 The Signal-to-Noise Ratio (SNR) Metric

The perceived tonal quality of an audio signal synthesised using sinusoidal or multi-harmonic WLS can be quantified by determining the ratio of RMS signal to noise amplitudes, or *signal-to-noise ratio* (SNR) [Moore, 1977]. SNR provides a simple and intuitive performance metric for evaluating and comparing phase mapping algorithms. To determine the noise magnitude we first define the phase mapping amplitude error sequence, $\varepsilon_a(n)$, thus:

$$\varepsilon_a(n) = y(n) - y_r(n) \tag{5.4.1}$$

where $y(n)$ denotes our synthesised signal sequence computed using a particular phase mapping algorithm (e.g. interpolated WLS) and $y_r(n)$ denotes an *ideal reference* signal sequence of the same frequency, amplitude and phase computed using full arithmetic precision. The RMS noise amplitude is obtained by taking the standard deviation of the error sequence, $\varepsilon_a(n)$, with respect to the mean over N_s samples, thus:

$$\sigma(\varepsilon(n),N_s)=\sqrt{\frac{1}{N_s}\sum_{n=0}^{N_s-1}[\varepsilon_a(n)-\mu(\varepsilon_a(n),N_s)]^2}$$

$$=\sqrt{\frac{1}{N_s}\sum_{n=0}^{N_s-1}[y(n)-y_r(n)]^2}\Bigg|_{\mu(\varepsilon_a(n),N_s)=0} \tag{5.4.2}$$

where $\mu(\varepsilon_a(n),N_s)$ and $\sigma(\varepsilon_a(n),N_s)$ respectively denote the mean and standard deviation of the amplitude error signal computed over N_s samples. For $\mu(\varepsilon_a(n),N_s)=0$ we have the form given by Moore [1977]. Hence, we define the signal-to-noise ratio, $\mathrm{SNR}(N_s)$, computed over N_s samples, thus:

$$\mathrm{SNR}(N_s)=20\log\left(\frac{\sigma(y(n),N_s)}{\sigma(\varepsilon_a(n),N_s)}\right) \tag{5.4.3}$$

Phase accumulating oscillators using simple (i.e. phase truncated) table lookup phase-mapping exhibit exponential growth in table length with improving SNR. However, interpolation and trigonometric techniques reduce lookup table growth for a given SNR specification. Quantisation noise corresponding to the sample word size and arithmetic round-off error define an upper-bound on SNR performance.

We now define the reference and synthesised sequences (i.e. $y_r(n)$ and $y(n)$) to support a simulated assessment of phase mapping algorithms using the SNR metric. We define the sinusoidal and multi-harmonic *reference* sequences, thus:

$$y_r(n)=\cos\left(2\pi\frac{\langle n\varphi\rangle_{2^M}}{2^M}\right)$$

$$y_r(n)=\sum_{k=1}^{N_h}A_k\cos\left(2\pi\frac{k\langle n\varphi\rangle_{2^M}}{2^M}\right) \tag{5.4.4}$$

where our normal parameter denotations apply.

For a given value of I and therefore wavetable length, we define the single-cycle *sinusoidal* or *multi-harmonic* wavetable vector, $\mathbf{T}$, according to Eqs. (4.1.2) and (4.1.3), respectively. There are three cases to consider for the definition of our synthesised signal sequences: interpolated phase mapping, rounded phase mapping and TIPM, observing that TIPM applies to sinusoidal synthesis only. Using the Lagrange interpolation expression of Eq. (5.2.2) we define the phase mapped sequence using order-N interpolation, thus:

$$y(n) = \sum_{k=0}^{N} \left[\mathbf{T}\left[\left\langle \phi_I(n) - \left\lfloor \frac{N-1}{2} \right\rfloor + k \right\rangle_{2^I} \right] \prod_{\substack{j=0 \\ j \neq k}}^{N} \left(\frac{\alpha(n) + \left\lfloor \frac{N-1}{2} \right\rfloor - j}{k-j} \right) \right] \quad (5.4.5)$$

where $\phi_I(n)$ and $\alpha(n)$ are defined by Eqs. (5.1.2) and (5.1.5), respectively. We also define the rounded phase mapping sequence according to Eq. (5.1.3), thus:

$$y(n) = \mathbf{T}\left[\left\langle \left\lfloor \frac{\phi(n)}{2^F} + 0.5 \right\rfloor \right\rangle_{2^I} \right] \quad (5.4.6)$$

We define the synthesised sinusoidal sequence using *trigonometric identity phase mapping* (TIPM) according to Eqs. (5.1.2), (5.1.4), (5.3.4), (5.3.6) and (5.3.7), thus:

$$y(n) = \mathbf{T}_2[\phi_I(n)]\mathbf{T}_3[\phi_F(n)] + \mathbf{T}_1[\phi_I(n)]\mathbf{T}_4[\phi_F(n)] \quad (5.4.7)$$

The sequences defined in Eqs. (5.4.4) through (5.4.7) underpin the SNR models presented in Appendix B.

5.4.2 Phase Increment and Phase Truncation Error

Nicholas *et al* [1987, 1988] derive analytical expressions for the number of lines in the phase-truncated sinusoidal signal spectrum, their amplitude and relative position as a function of M, φ, F and f_s. The numerical period, P, of $\phi(n)$ (i.e. the *minimum* value of P for which $\phi(n) = \phi(n+P)$) is fundamental to quantifying phase truncation

spectral errors and given by $P = \frac{2^M}{\gcd(\varphi, 2^M)}$, where $\gcd(\varphi, 2^M)$ denotes the greatest common divisor of φ and 2^M.

The authors represent the phase-truncated sinusoidal output sequence as $y_t(n) = \sin\left(\frac{2\pi}{2^M}\left[n\varphi - \varepsilon_p(n)\right]\right)$, where $\varepsilon_p(n)$ denotes a *phase error sequence* due to phase truncation and given by $\varepsilon_p(n) = \frac{2\pi}{2^M}\left(n\varphi - 2^F\left\lfloor\frac{n\varphi}{2^F}\right\rfloor\right) \in [0, \frac{2\pi}{2^I})$. $\varepsilon_p(n)$ is periodic with period $p_\varepsilon T$, where $p_\varepsilon \in [1, 2^F)$ and underpins the results presented. The principal result is that the number of spectral lines and their magnitude depend on φ through $\gcd(\varphi, 2^F)$ alone. As a consequence, values of φ that have the same value of $\gcd(\varphi, 2^F)$ cause the *number* of lines and their *respective amplitudes* to remain the same – only the *position* of each line in the spectrum changes. It is shown that the *number* of spectral lines is given by:

$$\Lambda = \frac{2^F}{\gcd(\varphi, 2^F)} - 1 \qquad (5.4.8)$$

and is a maximum when φ is odd and so $\gcd(\varphi, 2^F) = 1$, hence $\Lambda_{\max} = (2^F - 1)$. Values of φ which give $\gcd(\varphi, 2^F) = 2^F$ produce no phase truncation errors since $\Lambda = 0$ and so $\varepsilon_p(n) = 0$.

In the simulations whose results are presented in section (5.5), we choose values of φ which give $\gcd(\varphi, 2^F) = 1$ to maximise the number of phase-truncation error lines in the spectrum. This tends to maximise the energy in the error spectrum and hence by Parseval's theorem the corresponding energy in the amplitude error sequence ensuring a worst case simulated SNR figure. (Parseval's theorem relates the *total energy* of a DT sequence, $y(n)$, to the corresponding discrete Fourier transform (DFT), $Y(k)$, thus

$\sum_{n=0}^{N-1} y^2(n) = \frac{1}{N}\sum_{k=0}^{N-1}|Y(k)|^2$ [Ifeachor & Jervis, 2002]. Hence the energy of a DT amplitude sequence and its DFT are equivalent.)

Since Eq. (5.4.8) gives only the *number* of phase truncation spectral lines and not their amplitude distribution, we investigate the assumption that Λ_{max} corresponds with worst case SNR by simulating an example configuration over the permissible range of φ values and confirm approximate SNR invariance with φ. Figure (5.4.1) illustrates the simulated variation of SNR with φ for a 12-bit phase accumulator with 6-bit phase truncation indexing a 64 location sinusoidal wavetable. The relatively small phase accumulator word size enables all φ values to be simulated. The results confirm that the condition $\frac{2^F}{\gcd(\varphi, 2^F)} = 1$ yields maximum SNR bound only by computation round-off error, with minimum SNR for all odd φ. Figure (5.4.2) presents a corresponding theoretical prediction for the number of error spectrum lines, Λ, against φ according to Eq. (5.4.8). As expected, this corresponds with Figure (5.4.1) where $\Lambda = 0$ and $\Lambda = 2^F - 1$ yield maximum and minimum SNR, respectively.

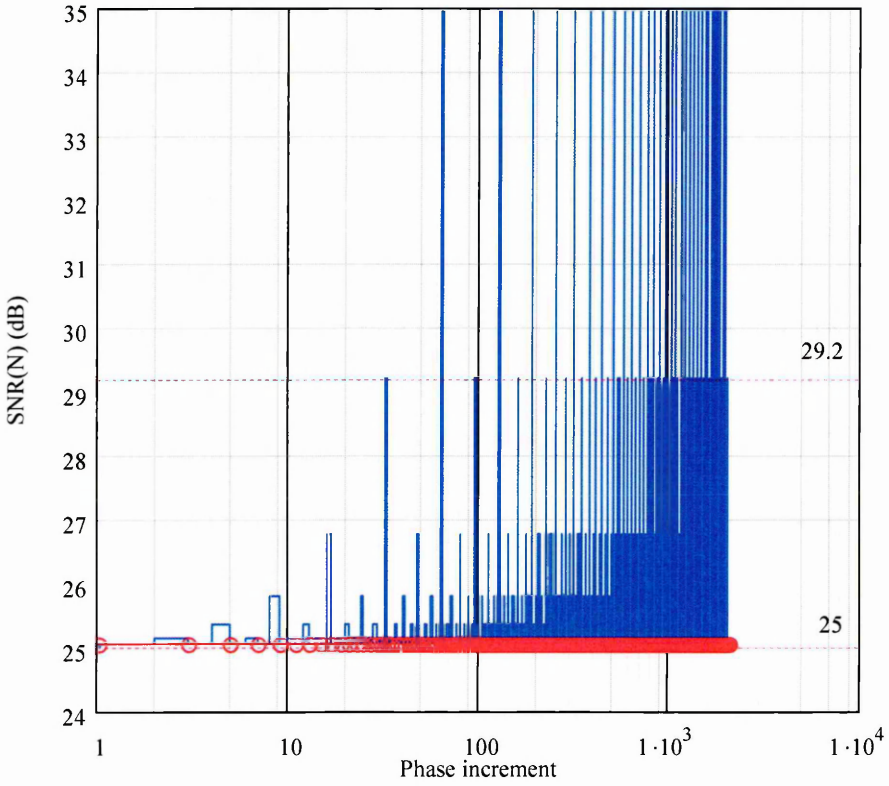


Figure (5.4.1): Variation of SNR with $\varphi \in [1, 2^M - 1]$ *for* $M = 12$ *and* $I = F = 6$. *Blue plot –* φ *ranging over all values on* $[1, 2^M - 1]$ *showing SNR maxima when* φ *is even. Red plot –* φ *ranging over all odd values on* $[1, 2^M - 1]$ *showing minimum SNR.*

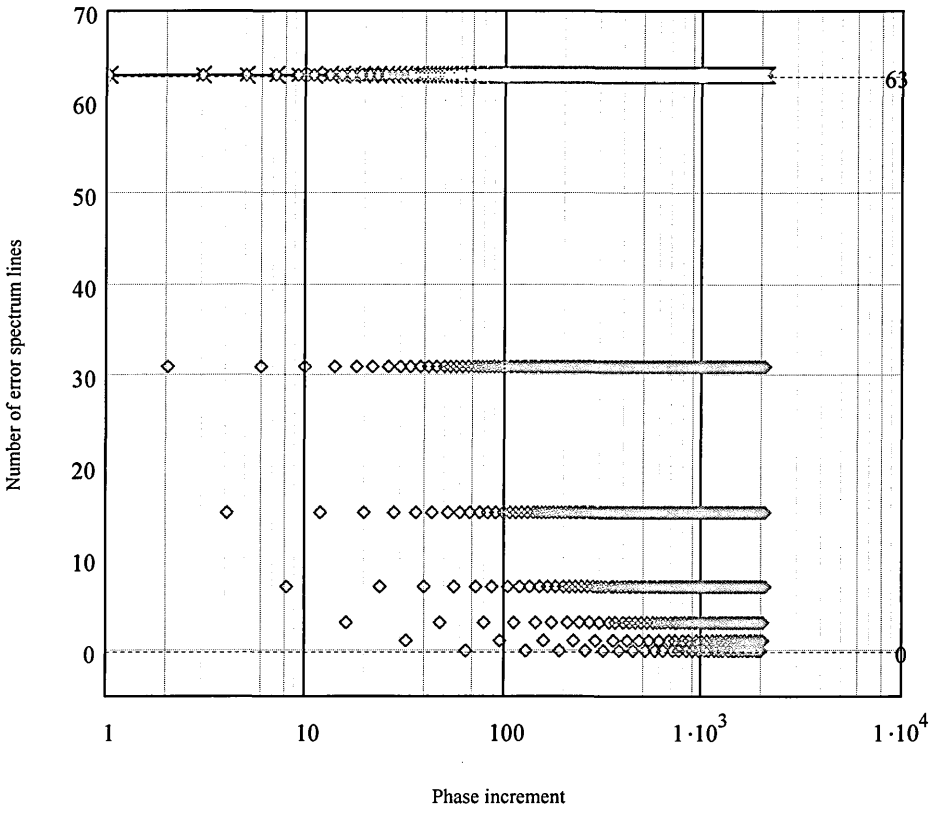


Figure (5.4.2): Variation of Λ *with* $\varphi \in [1, 2^M - 1]$, $M = 12$ *and* $I = 6$. *Green plot –* φ *ranging over all values on* $[1, 2^M - 1]$ *corresponding with local SNR peaks from Figure (5.4.1) when* Λ *deviates from the modal value of 63. Red plot –* φ *ranging over all odd values on* $[1, 2^M - 1]$. *Observe* Λ *is always a maximum for odd* φ. *(Plot lines have been omitted for clarity.)*

5.4.3 The Amplitude Error Spectrum

As an adjunct to the SNR(N_s) metric we compute the amplitude error spectrum by taking a windowed discrete Fourier transform (DFT) of the amplitude error sequence given by Eq. (5.4.1) to illustrate the frequency domain characteristics of phase truncation noise. Computing the DFT necessarily forces a finite duration analysis record or "window" of N_sT seconds. In the simplest case, the window is effectively a rectangular function of width N_s samples. The DFT periodically extends this analysis window causing signals whose period is not a sub-multiple of N_sT to exhibit discontinuities at the window boundaries. These discontinuities cause *spectral leakage* in the DFT output spectrum which manifest as spectral lines not present in the original signal. To mitigate spectral leakage we apply a non-rectangular window to the analysis record. The window function applies a multiplicative weighting to the analysis record which tapers to zero at the record boundaries and so reduces the contribution from the periodic extension discontinuities. We use the Hamming window function as this provides good sidelobe suppression of -43 dB and a sidelobe roll-off of -6 dB/octave [Harris, 1978]. Figure (5.4.3) illustrates the Hamming window for $N_s = 1024$ together with its frequency response for $N_s = 1024$ and $N_s = 65536$. As N_s increases, the main lobe width decreases and the highest sidelobe level remains the same. However, since the density of sidelobes increases with N_s, the constant sidelobe roll-off causes suppression of higher sidelobes to improve with increasing N_s.

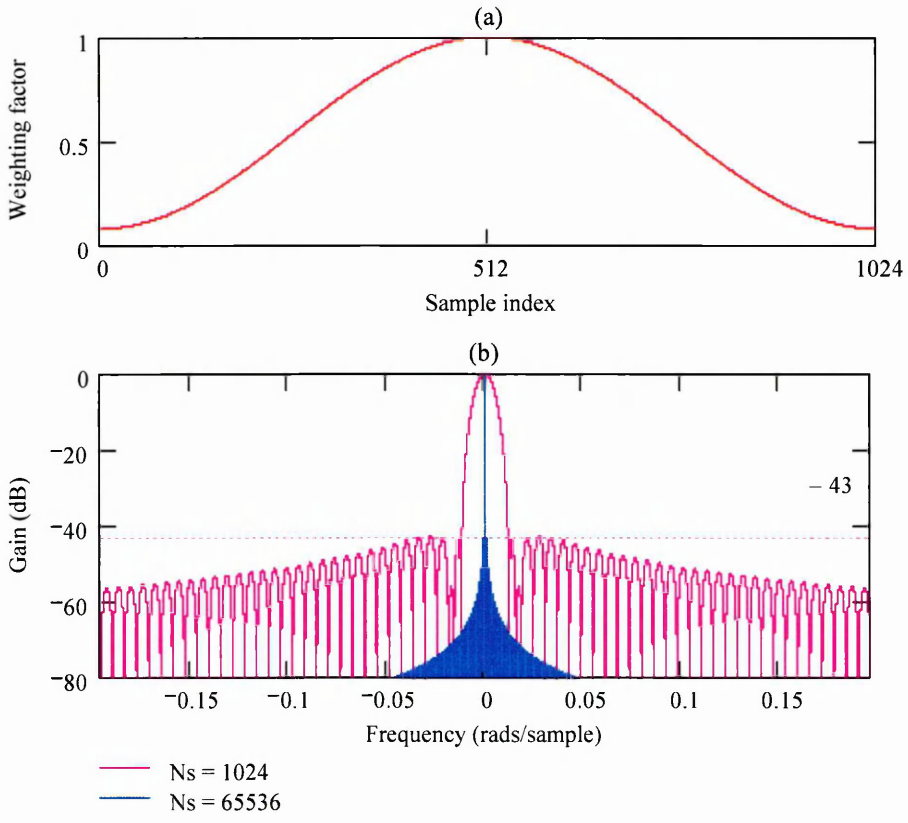


Figure (5.4.3): (a) - Time domain response of the Hamming window for $N_s = 1024$. *(b) - Frequency response of the Hamming window for* $N_s = 1024$ and $N_s = 65536$. *A particular bin centre is normalised to zero frequency on the frequency axis.*

When computing the phase truncated signal spectrum, the DFT bin width for a record size of N_s samples is given by [Orfanidis, 1996]:

$$f_b = \frac{f_s}{N_s} \tag{5.4.9}$$

The spectral resolution, Δf, [Harris, 1978] must take account of the *equivalent noise bandwidth* (ENBW), β, of the window function expressed in bins and is defined thus:

$$\Delta f = \beta\left(\frac{f_s}{N_s}\right) \tag{5.4.10}$$

For the Hamming window we have $\beta = 1.36$.

5.4.4 Defining the Wavetable Spectrum

To simulate phase truncation effects and interpolated phase mapping WLS with non-sinusoidal wavetables, we define piecewise-linear spectrum models of musical signals to construct wavetable spectra using results presented in Borch & Sundberg [2002]. This paper presents analyses of several music types using *long-term average spectrum* (LTAS) analysis presented in Jansson & Sundberg [1976] to measure an average spectral "signature" for a particular type of music. Both popular and classical music types have been analysed and the results approximated here by piecewise-linear spectral envelopes allowing the computation of multi-harmonic wavetables using Eq. (4.1.3).

We first define a generalised piecewise-linear harmonic spectrum, A_k, comprising N_h harmonics with $k \in [1, N_h]$, fundamental frequency, f_0 and a single breakpoint harmonic index, k_b (with corresponding frequency $k_b f_0$). Two variables, r_1 and r_2, define the spectrum slopes before and after the breakpoint harmonic, respectively. We have:

$$A_k = \begin{cases} A\left(\frac{1}{k}\right)^{r_1} & k \in [1, k_b) \\ A\left(\frac{1}{k_b}\right)^{r_1 - r_2}\left(\frac{1}{k}\right)^{r_2} & k \in [k_b, N_h] \end{cases} \qquad (5.4.11)$$

where A represents the fundamental amplitude and A_k represents the k^{th} harmonic amplitude. Using the results from Borch & Sundberg [2002] and Eq. (5.4.11) we define several piecewise-linear spectra which approximate various musical types. Our objective is to define wavetable spectra which capture the essence and diversity of real musical signals and hence underpin simulation of interpolated WLS SNR performance. These spectral envelopes are illustrated in Figures (5.4.4) through (5.4.6) and define the amplitude coefficients, A_k, which generate reference wavetables using Eq. (4.1.3) with

$\Phi_k = 0$. We base these spectra on two fundamental frequencies – $f_0 = 16.35$ Hz (C0) and $f_0 = 130.81$ Hz (C3) with corresponding phase increments $\varphi = 5715$ and $\varphi = 45721$, assuming $f_s = 48\,\text{kHz}$ and $M = 24$. Both of these φ values satisfy the condition $\gcd(\varphi, 2^F) = 1$ presented in section (5.4.2), thereby ensuring a maximum number of error spectrum lines according to $\Lambda_{\max} = (2^F - 1)$ and hence worst case SNR.

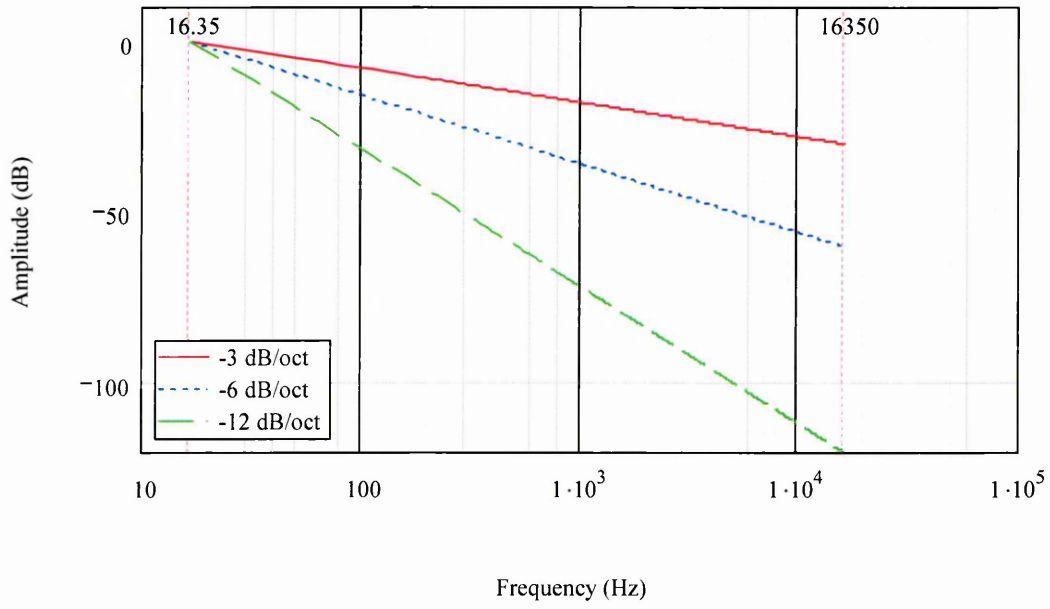


Figure (5.4.4): Single slope spectra ranging over 1000 harmonics covering a bandwidth of 16.35 Hz to 16,350 Hz which approximates various popular music LTAS reported in Borch & Sundberg [2002].

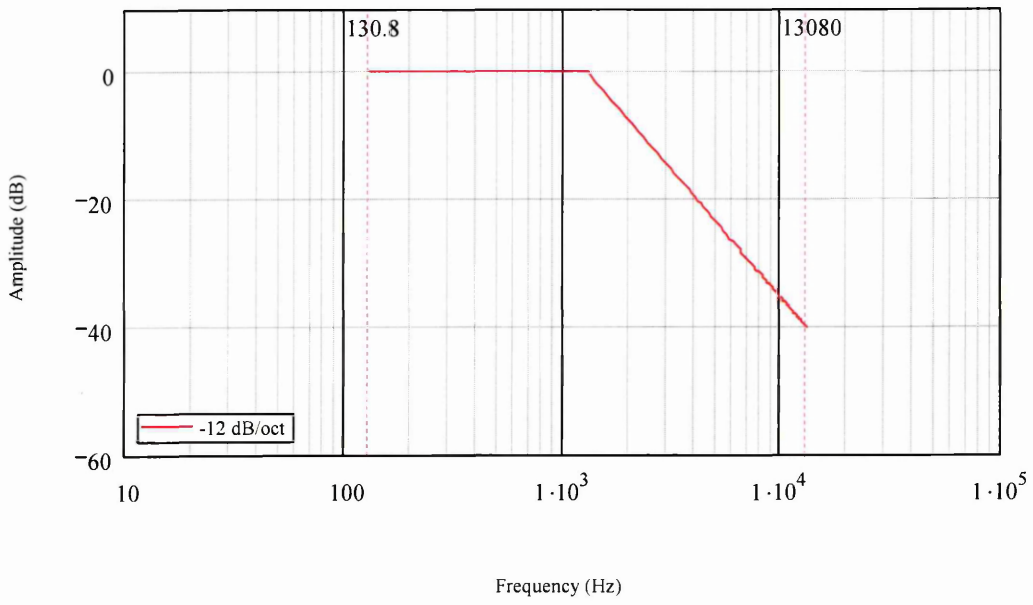


Figure (5.4.5): Piecewise-linear spectrum ranging over 100 harmonics, covering a bandwidth of 130.81 Hz to 13,081 Hz which approximates various classical music LTAS reported in [Borch & Sundberg, 2002].

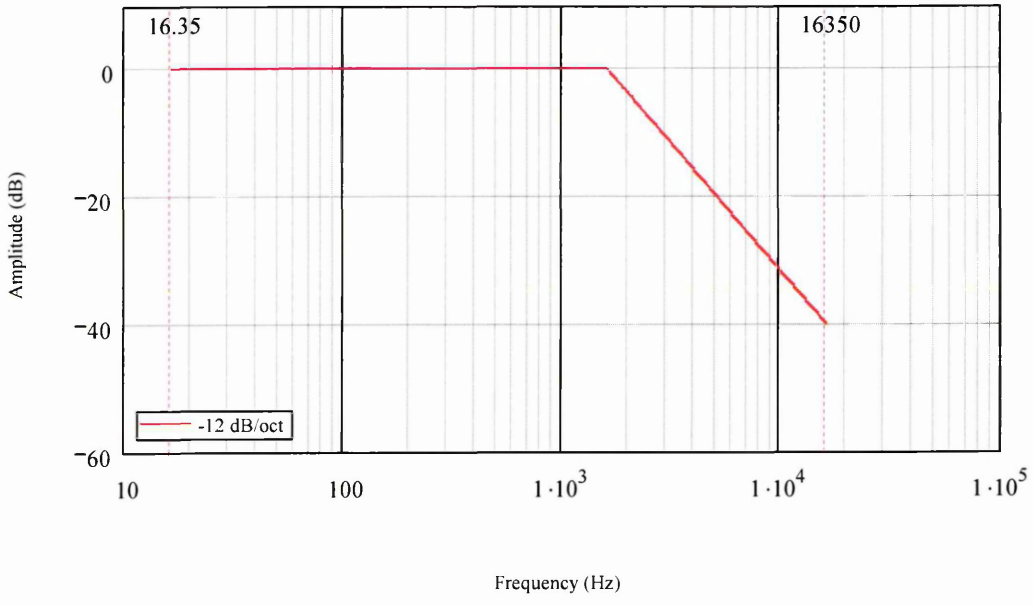


Figure (5.4.6): Piecewise-linear spectrum ranging over 1000 harmonics covering a bandwidth of 16.35 Hz to 16,350 Hz.

5.4.5 Simulation Record Length

Ideally, we require the fundamental of the simulated synthesised signal to lie precisely at a bin centre implying an output frequency, f, which is a precise integer multiple of f_b, or $f = kf_b$, with k a positive integer. Setting $N_s = 2^M$ ensures that φ is always *precisely* equivalent to the DFT bin number and also accommodates the maximum numerical period of the phase sequence when $(\varphi, 2^M) = 1$. However, this requires a DFT length on the order of 2^{24} samples which is impractical for reasonable simulation execution times and vector sizes. For $N_s < 2^M$, the ratio $\frac{f}{f_b}$ takes on fractional values and bin leakage results. We define the *fractional bin tuning error* metric, $\varepsilon_b(N_s)$, as the fractional component of $\frac{f}{f_b}$, thus:

$$\varepsilon_b(N_s) = \left(\frac{\varphi N_s}{2^M} - \left\lfloor \frac{\varphi N_s}{2^M} \right\rfloor \right) \tag{5.4.12}$$

and we select N_s for particular values of φ and M so as to minimise $\varepsilon_b(N_s)$ on a range of acceptable N_s values.

Having defined our fundamental test frequencies and therefore corresponding φ values in section (5.4.4), we now proceed to determine N_s given a particular error bound on $\varepsilon_b(N_s)$. The function $\varepsilon_b(N_s)$ is sawtooth in nature with a distribution of sharply defined minima over a range of N_s for particular values of φ and M. The localised minima are exposed with a logarithmic plot as illustrated in Figures (5.4.7) and (5.4.8), which show the variation of $\varepsilon_b(N_s)$ with N_s over the range $N_s \in [2^{10}, 2^{16}]$. ($N_s = 2^{16}$ represents the limit of DFT record length for acceptable execution time on the available computer platform.) Inspecting Figures (5.4.7) and (5.4.8) we have determined values of

N_s yielding minimum $\varepsilon_b(N_s)$ over a range of N_s values and hence maximum DFT lengths consistent with acceptable simulation times. For $f_s = 48\,\text{kHz}$ the chosen record length values $N_s = 49906$ and $N_s = 56143$ provide corresponding DFT frequency resolutions of 1.3 Hz and 1.2 Hz, respectively (i.e. ≈ 1 Hz), assuming a Hamming analysis window. We use a non-radix FFT algorithm available in Mathcad to compute the DFT spectra presented in section (5.5).

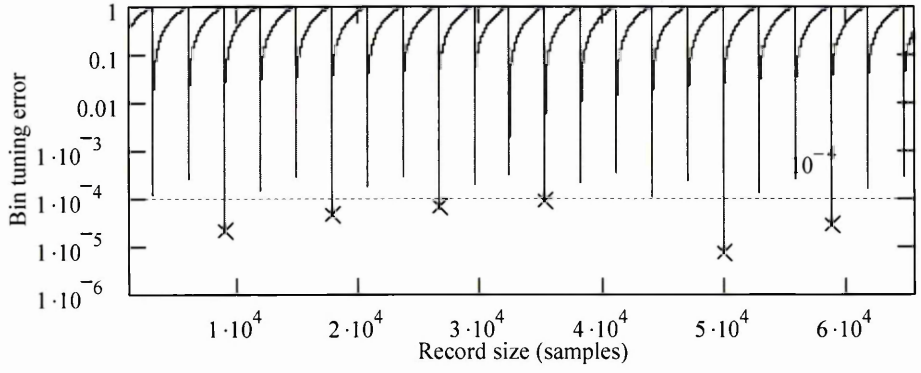


Figure (5.4.7): Behaviour of $\varepsilon_b(N_s)$ *over* N_s *for* $\varphi = 5715$ *and* $M = 24$*, where the marker* × *denotes* $\varepsilon_b(N_s) < 10^{-4}$*.* $N_s = 49906$ *gives minimum* $\varepsilon_b(N_s)$ *on* $N_s \in [2^{10}, 2^{16}]$.

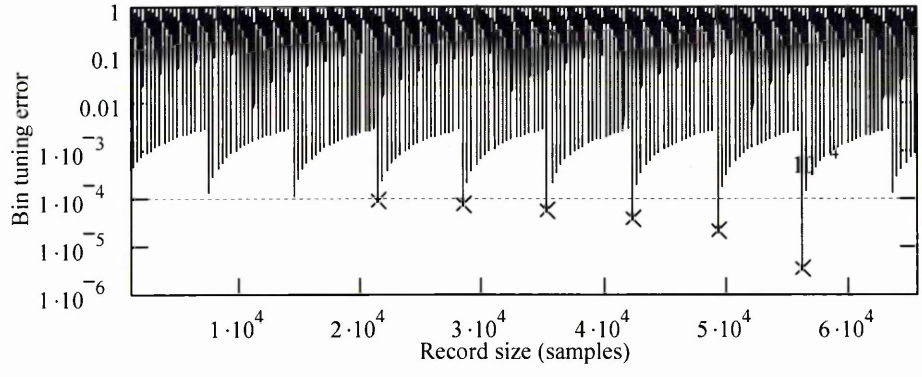


Figure (5.4.8): Behaviour of $\varepsilon_b(N_s)$ *over* N_s *for* $\varphi = 45721$ *and* $M = 24$*, where the marker* × *denotes* $\varepsilon_b(N_s) < 10^{-4}$*.* $N_s = 56143$ *gives minimum* $\varepsilon_b(N_s)$ *on* $N_s \in [2^{10}, 2^{16}]$.

5.5 Simulation Results

5.5.1 Introduction

In this section we present simulation results for six phase-mapping algorithms and define the following acronyms and corresponding colour codes to simplify annotation:

- TPM – *truncated* phase mapping — colour code red
- RPM – *rounded* phase mapping — colour code brown
- LIPM – *linear* interpolation phase mapping — colour code blue
- QIPM – *quadratic* interpolation phase mapping — colour code green
- CIPM – *cubic* interpolation phase mapping — colour code magenta
- TIPM – *trigonometric identity* phase mapping — colour code black

TPM and RPM are both forms of zero-order interpolation, with RPM utilising phase fraction information to *round* the phase integer component to the nearest integer value. LIPM, QIPM and CIPM effect Lagrange polynomial interpolation of the wavetable indexing operation with interpolation order $N = 1, 2, 3$, respectively. We are not concerned with the computational advantage afforded by the Newton interpolation polynomial representation and seek only to present a relative qualitative assessment of each interpolation algorithm applied to the phase mapping problem. In particular, the Lagrange polynomial is likely to give a larger (i.e. worst case) amplitude error estimate due to the increased number of multiplication operations and hence corresponding round-off errors in the β coefficient computations (see Eq. (5.2.3)). TIPM represents the phase mapping technique presented in section 5.3 which uses the angle summation

trigonometric identity to effect optimal sinusoidal phase mapping with reduced wavetable memory compared to the brute force approach.

SNR and amplitude error spectrum simulations use the Mathcad programs presented and documented in Appendix B. We adopt the definition of SNR peculiar to phase mapping accuracy given by Eq. (5.4.3) which returns *positive* values for signal power *greater* than noise power (i.e. increasingly positive values indicate *improving* performance). The amplitude axes of amplitude error spectrum plots indicate *absolute* noise power relative to a unit-amplitude reference signal (i.e. increasingly *negative* values indicate *improving* performance). As an adjunct to amplitude error spectra we also consider the *spurious-free dynamic range* metric (SFDR), defined as the *largest* spectral component within the Nyquist region of the normalised amplitude error spectrum.

Mathcad simulation programs model fixed-point arithmetic operations and wavetable tabulations using the quantiser function given by Eq. (4.1.5). Interpolation is modelled by the Lagrange polynomial defined by Eq. (5.2.2). Multiplication operations are rounded to double fixed-point precision relative to the input operands with subsequent accumulation operations performed to full fixed-point precision (e.g. two 16-bit multiplier operands produce a 32-bit result which is accumulated to 32-bit resolution). The interpolated output sample computed from the multiply-accumulate operation implicit within Eq. (5.2.2), is rounded to the appropriate word size prior to SNR computation (e.g. 32-bit multiply-accumulate results are rounded to 16-bit precision).

5.5.2 Sinusoidal Phase-Mapping – Non-Truncated Phase Fraction

Figures (5.5.1) through (5.5.3) illustrate composite simulation plots of SNR variation with I (i.e. wavetable length $L = 2^I$) using a single sinusoid wavetable, the six interpolation algorithms and three arithmetic quantisation levels: full floating-point precision, 24-bit fixed-point precision and 16-bit fixed-point precision. In all cases it is evident that the SNR value obtained with TIPM is independent of I and bound by quantisation noise only. The ≈300 dB SNR upper-bound evident in Figure (5.5.1) represents the "computational noise" ceiling corresponding to the floating point arithmetic precision of the Mathcad simulation environment.

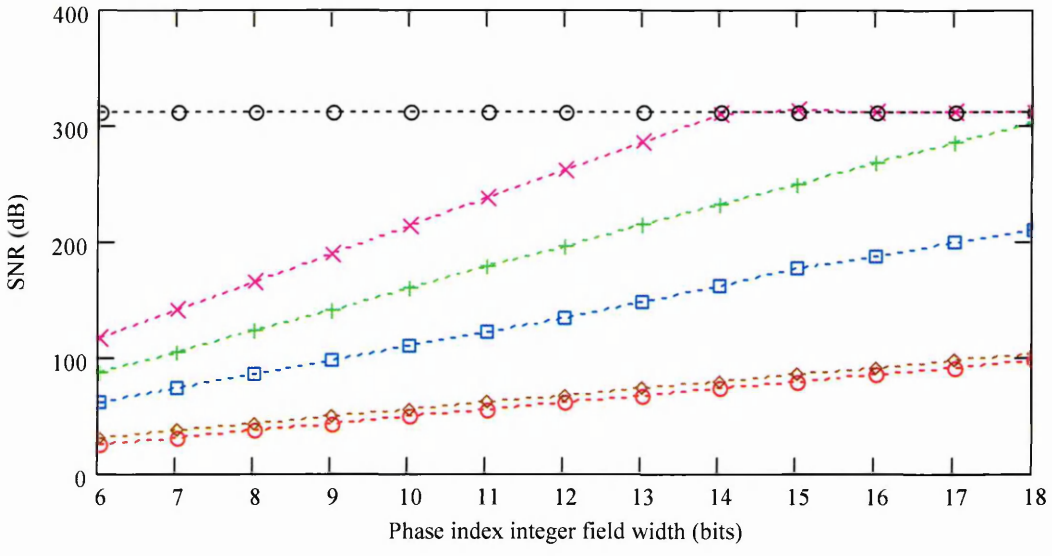


Figure (5.5.1): SNR variation with I for interpolated sinusoidal phase mapping, with $M = 24$, $\varphi = 5715$, $N_s = 49906$ and full precision arithmetic.

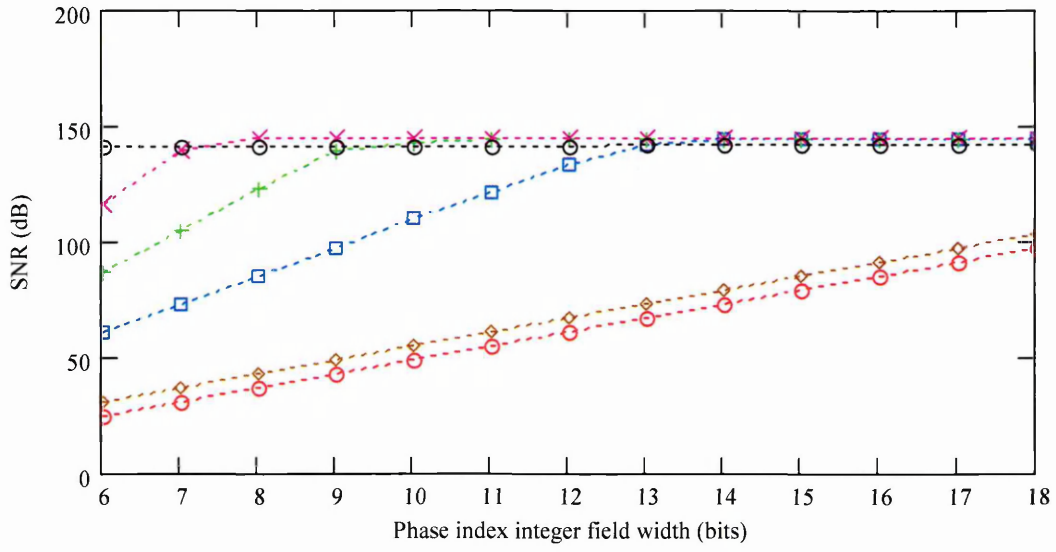


Figure (5.5.2): SNR variation with I for interpolated sinusoidal phase mapping, with $M = 24$, $\varphi = 5715$, $N_s = 49906$ and 24-bit arithmetic.

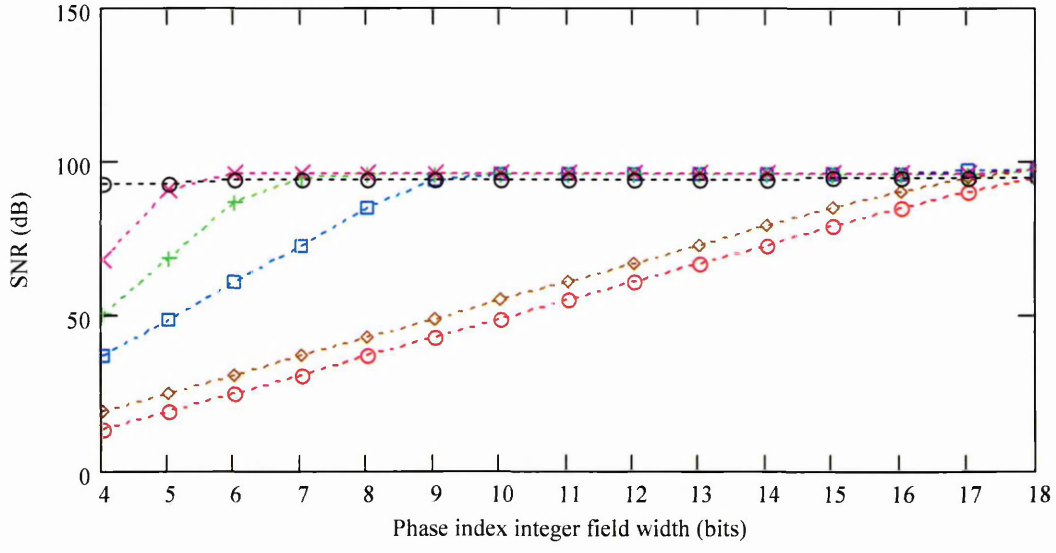


Figure (5.5.3): SNR variation with I for interpolated sinusoidal phase mapping, with $M = 24$, $\varphi = 5715$, $N_s = 49906$ and 16-bit arithmetic.

Figure (5.5.2) shows that SNR levels consistent with a 24-bit quantisation noise ceiling, where Eq. (4.1.6) gives $\text{SQNR} \approx 146\,\text{dB}$, require wavetable lengths of 256, 512 and 8192 samples using cubic, quadratic and linear interpolation, respectively. Similarly,

SNR levels consistent with a 16-bit quantisation noise ceiling, where Eq. (4.1.6) gives $SQNR \approx 98\,dB$, require wavetable lengths of 64, 128 and 512 samples using cubic, quadratic and linear interpolation, respectively. SNR expressed in dB, improves linearly with I and is always constrained by the SQNR upper-bound. Figures (5.5.4) and (5.5.5) illustrate the behaviour of SNR with interpolation order over the range $N \in [0, 10]$ with phase integer field widths $I = 8$ and $I = 12$, respectively. For a given arithmetic word size and hence quantisation interval, SNR increases monotonically with N and is constrained by the SQNR ceiling for large N. The rate at which SNR approaches the SQNR ceiling increases with I and hence wavetable length.

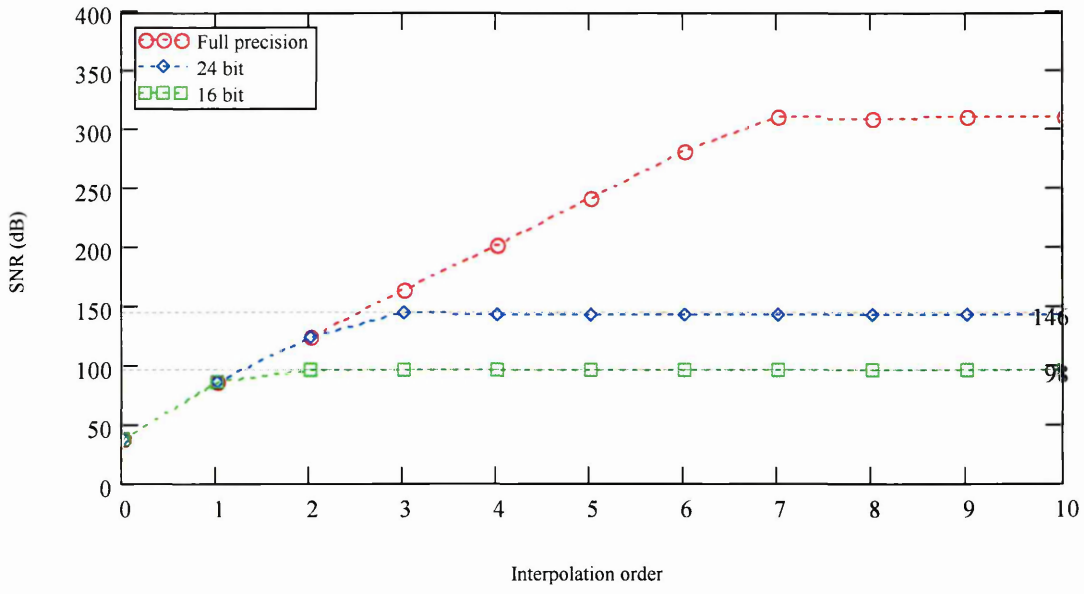


Figure (5.5.4): SNR variation with interpolation order $N \in [0, 10]$ using full precision, 24-bit and 16-bit arithmetic and a 256 sample wavetable ($I = 8$).

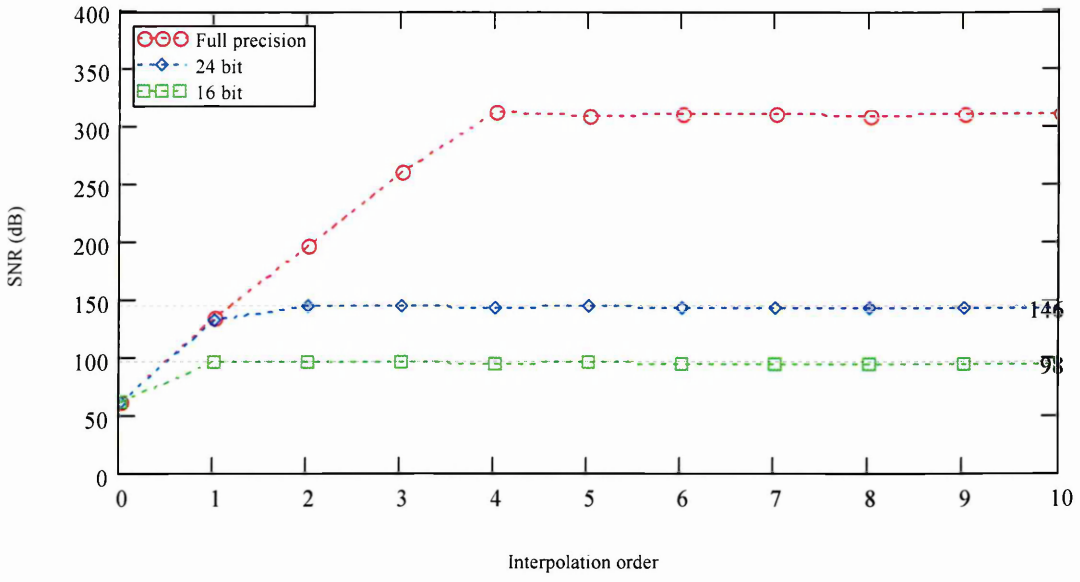


Figure (5.5.5): SNR variation with interpolation order $N \in [0, 10]$ *using full precision, 24-bit and 16-bit arithmetic and a 4096 sample wavetable (*$I = 12$*).*

The results confirm that TIPM yields SNR bound by sample quantisation noise and is independent of the partition between I and F, as expected. TIPM requires two multiplication operations and imposes a lookup table memory overhead of $2^{\frac{M}{2}+2}$ samples, irrespective of the SNR requirement. LIPM yields a SNR which is a function of I alone and bound by sample quantisation noise. LIPM requires a single multiplication operation and imposes a lookup table memory overhead of 2^{I+1} samples, assuming two lookup tables provide the sample and first-order difference operands within a single read cycle as illustrated in the process model of Figure (3.3.3). We conclude that TIPM and LIPM are the preferred phase mapping algorithms for sinusoidal wavetables based on the SNR metric. In general, TIPM is preferred over LIPM when we require SNR values above the 24-bit SQNR bound or optimal phase control resolution of $\frac{2\pi}{2^M}$ radians. For computer music applications which are

constrained by human auditory perception requirements, LIPM provides an optimal solution to the sinusoidal phase mapping problem.

5.5.3 Sinusoidal Phase Mapping – Truncated Phase Fraction

We continue our investigation of sinusoidal wavetable phase mapping by simulating the effect of a truncated phase fraction field on SNR. Section (5.1.2) reviewed the partitioning of an M-bit phase word into integer and fraction fields with the fraction field truncated by R bits. For QIPM, CIPM and higher order interpolation algorithms, it is evident from simulations whose results are not presented here that a *single* bit truncation of the phase fraction field (i.e. $R = 1$) causes a large reduction in SNR whose magnitude increases with I. Figures (5.5.6) and (5.5.7) illustrate SNR variation with R for the TIPM and LIPM algorithms, with $I = 12$ (i.e. $I = \frac{M}{2}$) for the TIPM simulation and I varying over the range $I \in [6, 12]$ for the LIPM simulation.

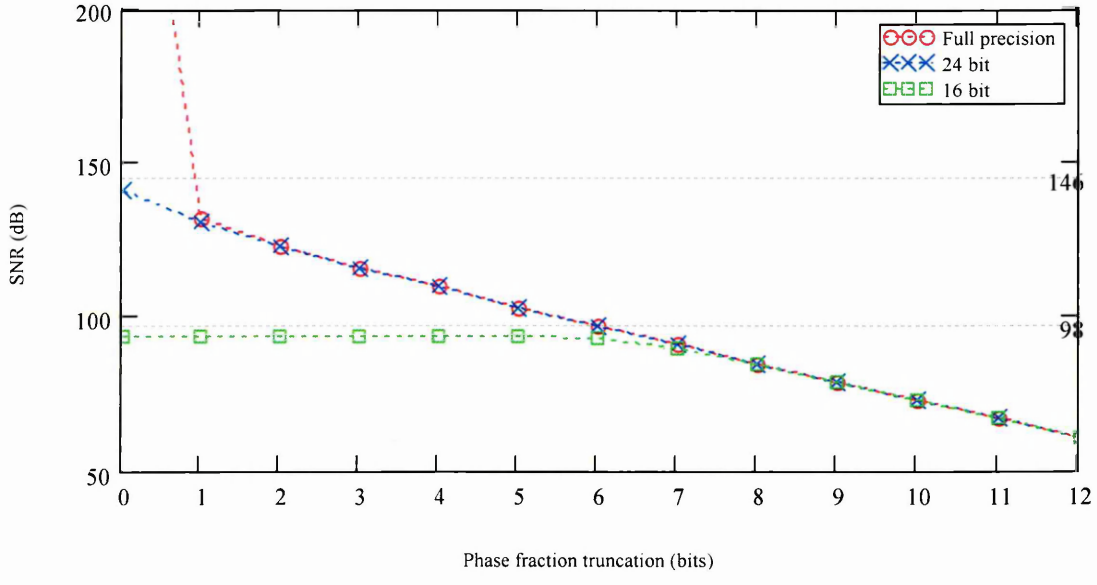


Figure (5.5.6): SNR as a function of R for TIPM using full precision, 24-bit and 16-bit arithmetic with $M = 24$, $I = 12$, $\varphi = 5715$, $N_s = 49906$ *and* $f_s = 48\,\text{kHz}$.

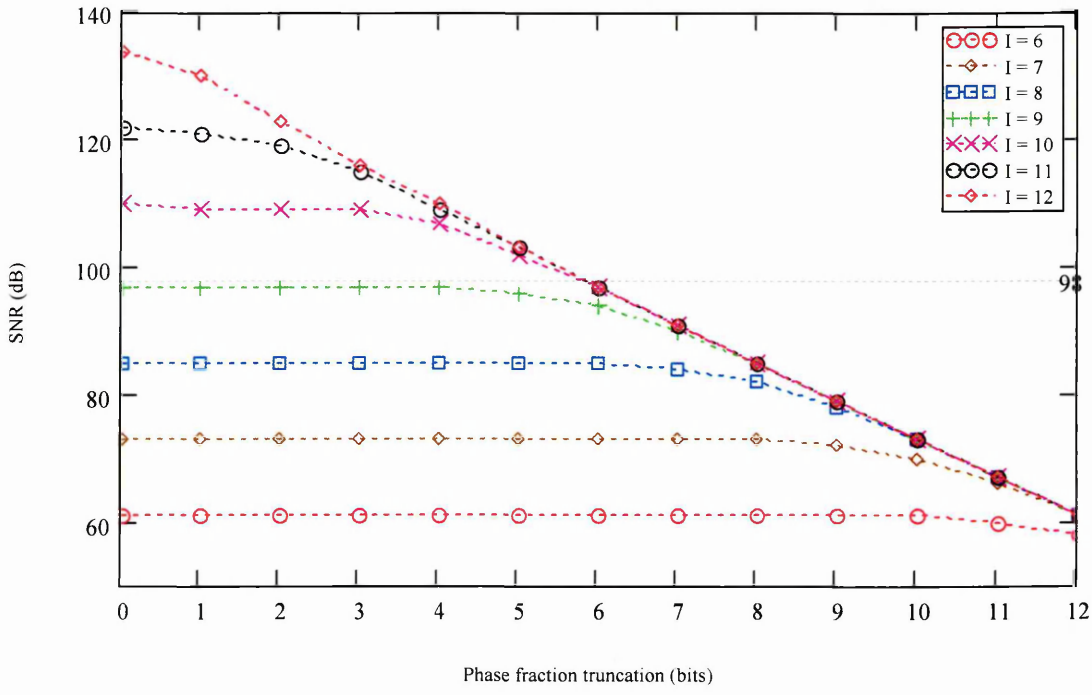


Figure (5.5.7): SNR as a function of R for LIPM using full precision arithmetic with $M = 24$, $I \in [6, 12]$, $\varphi = 5715$, $N_s = 49906$ and $f_s = 48\,\text{kHz}$.

From Figure (5.5.6) it is evident that SNR with TIPM is extremely sensitive to phase fraction truncation. A single bit truncation of the phase fraction field (i.e. $R = 1$) reduces SNR from the full precision computation noise ceiling (> 300 dB) to approximately 133 dB (i.e. 15 dB below the 24-bit SQNR ceiling). For 24-bit arithmetic, SNR falls with increasing R at an initial rate of 10 dB per bit reducing to 6 dB per bit for large R. For 16-bit arithmetic, SNR stays constant at the SQNR ceiling for $R \in [0, 6]$, thereafter falling at the same rate as the 24-bit case with increasing R. We conclude that with $M = 24$, $I = 12$ and 16-bit arithmetic, the 6 least significant bits of the phase fraction field are superfluous and can be truncated with no loss of SNR. This gives a total lookup table memory overhead of $2(2^{12}) + 2(2^{6}) = 8320$ 16-bit words. However, in this particular example we have $I + F = 18$ and by setting $I = F$ as discussed in section (5.3.2), we can reduce the lookup table memory overhead to $4(2^{9}) = 2048$ 16-bit words,

with no loss of SNR. In general, with reducing arithmetic word size and $R > 0$ optimum memory utilisation is evident when $I = F = (M - R)$, giving a total memory overhead of $4\left(2^{\frac{M-R}{2}}\right)$ words, assuming $M - R$ is divisible by two.

Figure (5.5.7) shows that LIPM with $I \leq 10$ exhibits SNR invariance with increasing R up to a "knee point" after which SNR falls linearly with increasing R. The initial SNR value ($R = 0$) is proportional to I, increasing by 12 dB for each bit increment in I. Decreasing I yields wider SNR invariance with R. For $I \geq 12$ there is essentially no SNR invariance region and SNR reduces steadily at 6 dB for each unit increase in R. We conclude that LIPM with $I \geq 12$ supports a well-behaved trade-off between SNR and phase fraction field width, F. This property finds utility in VLSI implementations where we wish to optimise the interpolation multiplier operand width and therefore gate count for a given SNR specification.

5.5.4 Sinusoidal Phase Mapping – Amplitude Error Spectra

Figure (5.5.8) illustrates a composite plot of simulated amplitude error spectra for the six interpolated phase mapping techniques using the Mathcad program presented in Appendix B with $M = 24$, $I = F = 12$ and $f_s = 48\,\text{kHz}$. We see a steady reduction in the amplitude of spurious spectral components and hence improving SFDR with increasing interpolation order. SFDR ranges from -62 dB for TPM, through -135 dB for LIPM to -325 dB for TIPM. The RPM error spectrum exhibits a suppressed component at the fundamental frequency in contrast to the TPM error spectrum where a residual fundamental component is evident. The two spectra are otherwise very similar, which is expected given the constant 6 dB difference in the corresponding SNR values, irrespective of I. Figure (5.5.9) illustrates the variation in SFDR with $I \in [6, 18]$

indicating an essentially constant SFDR performance with TIPM and linearly improving SFDR performance with our five interpolating phase mapping algorithms.

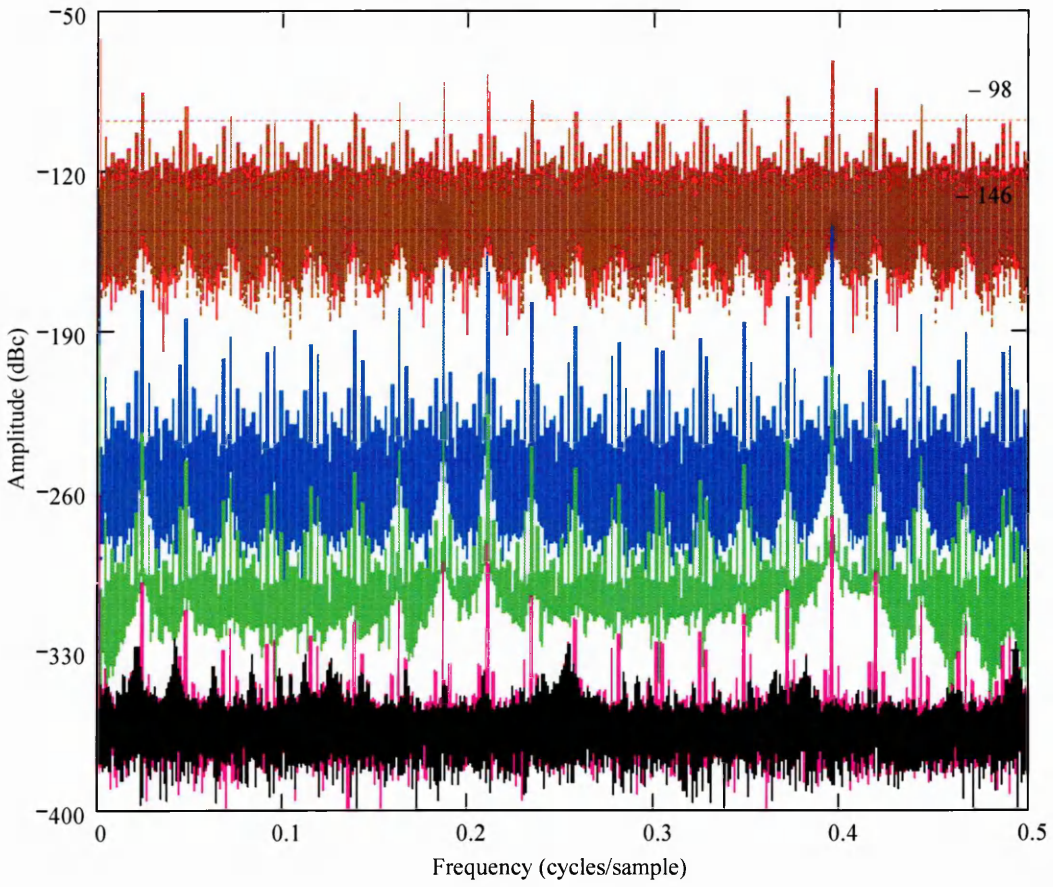


Figure (5.5.8): Composite amplitude error spectra for the six interpolated phase mapping techniques using $\varphi = 5715$ *(i.e.* $\approx$ *0.00034 cycles/sample), with* $M = 24$, $I = F = 12$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic. (RPM error spectrum is shown dashed to illustrate similarity with the underlying TPM error spectrum.)*

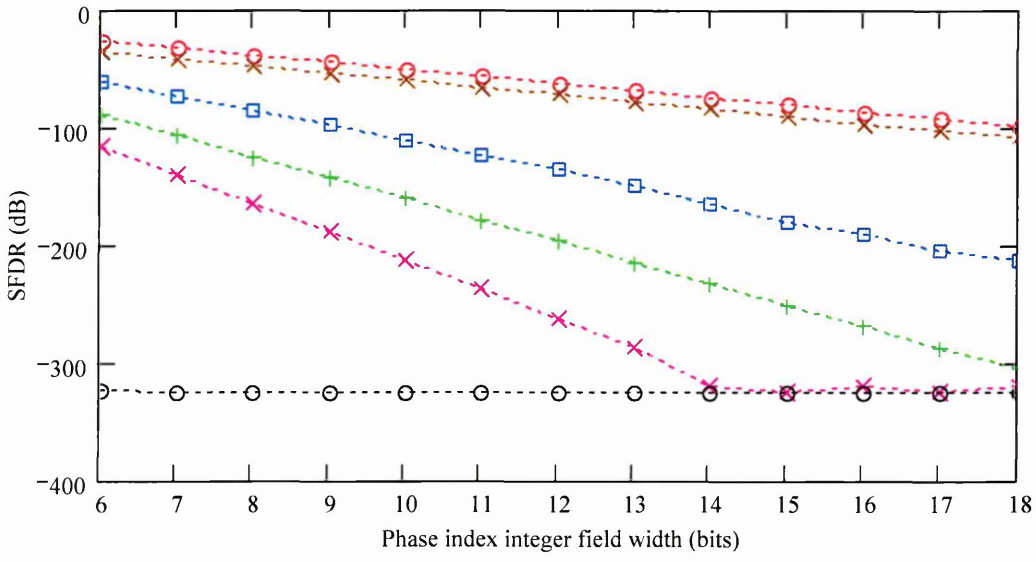


Figure (5.5.9): SFDR variation with $I \in [6, 18]$ *for six interpolated phase mapping techniques using* $\varphi = 5715$ *(i.e.* ≈ 0.00034 *cycles/sample), with* $M = 24$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*

5.5.5 Multi-Harmonic Phase Mapping

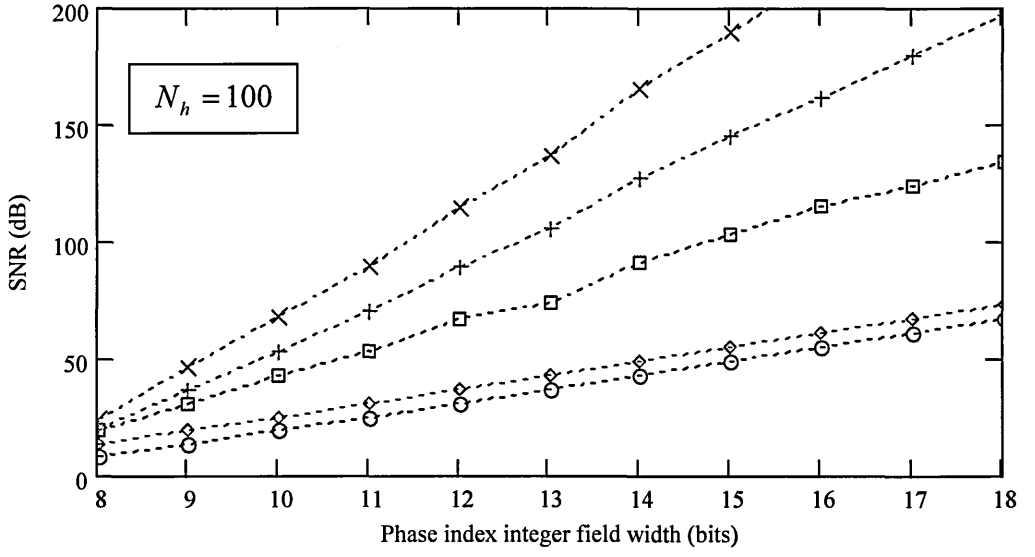
Figures (5.5.10) through (5.5.25) illustrate simulation results for interpolated multi-harmonic WLS assessing five interpolation algorithms with three arithmetic quantisation levels. We organise our simulations into four categories whose respective parameters are summarised in Table (5.5.1) and which comprise:

- SNR variation with I
- Amplitude error spectra across the Nyquist frequency range
- SFDR variation with I
- SNR variation with wavetable spectrum roll-off slope displayed as a contour plot with contours defining loci of constant SNR

We simulate the TPM, RPM, LIPM, QIPM and CIPM interpolation algorithms using full floating-point precision, 24-bit fixed-point precision and 16-bit fixed-point precision. Full precision simulations serve as a reference point providing baseline data free from quantisation error. The wavetable used in a particular simulation tabulates a single period, multi-harmonic waveform computed using Eq. (4.1.3) from an amplitude spectrum specification defined by the number of harmonics and the harmonic amplitude envelope profile according to Eq. (5.4.11). Our objective is to determine an interpolation order and corresponding wavetable length which provides an SNR value comparable with the SQNR of typical audio DSP environments (i.e. 16 and 24-bit fixed-point).

Figure	Type	Wavetable Spectrum	N_h	I	φ	N_s
(5.5.10a)	SNR - I	Fig. (5.4.4) -3 db/octave	100	[8, 18]	5715	49906
(5.5.10b)	SNR - I		1000	[11, 18]		
(5.5.11)	Error spectrum		100	12		
(5.5.12)	SFDR - I		100	[8, 18]		
(5.5.13a)	SNR - I	Fig. (5.4.4) -6 db/octave	100	[8, 18]	5715	49906
(5.5.13b)	SNR - I		1000	[11, 18]		
(5.5.14)	Error spectrum		100	12		
(5.5.15)	SFDR - I		100	[8, 18]		
(5.5.16a)	SNR - I	Fig. (5.4.4) -12 db/octave	100	[8, 18]	5715	49906
(5.5.16b)	SNR - I		1000	[11, 18]		
(5.5.17)	Error spectrum		100	12		
(5.5.18)	SFDR - I		100	[8, 18]		
(5.5.19a)	SNR - I	Fig. (5.4.5) -12 db/octave low-pass	100	[8, 18]	45721	56143
(5.5.19b)	SNR - I	Fig. (5.4.6) -12 db/octave low-pass	1000	[11, 18]	5715	49906
(5.5.20)	Error spectrum	Fig. (5.4.5) -12 db/octave low-pass	100	12	5715	49906
(5.5.21)	SFDR - I		100	[8, 18]	5715	49906
(5.5.22)	I - roll-off TPM	Single slope spectra according to Eq. (5.4.11) with roll-off ranging from 0 to -24 dB/octave	100	[8, 18]	5715	49906
(5.5.23)	I - roll-off LIPM					
(5.5.24)	I - roll-off QIPM					
(5.5.25)	I - roll-off CIPM					

Table (5.5.1): Simulation parameters and multi-harmonic wavetable characteristics supporting the performance assessment of interpolated phase mapping.



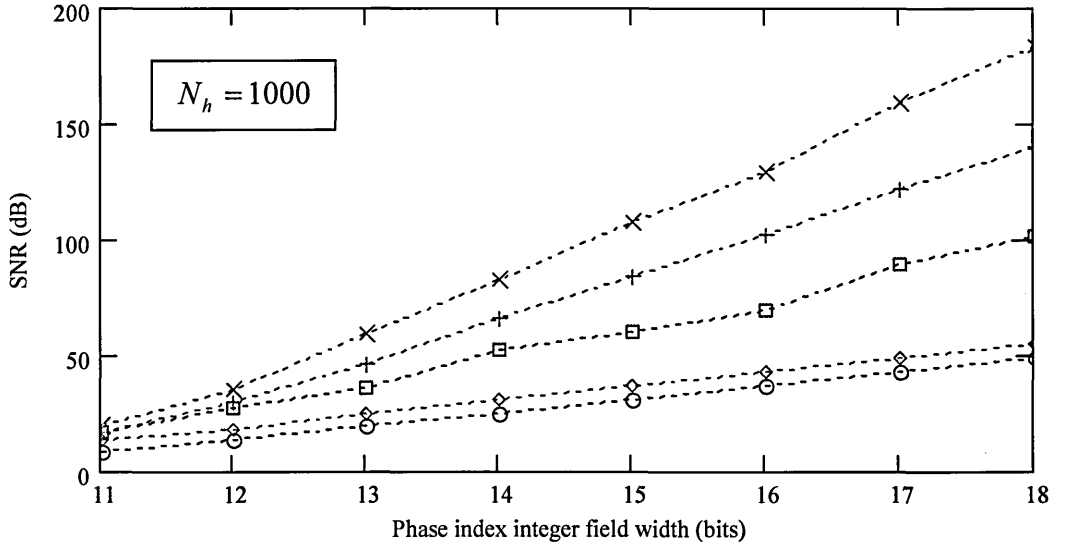


Figure (5.5.10): SNR variation with I for two N_h values. The phase mapping wavetable tabulates a multi-harmonic signal with -3 dB/octave spectrum as illustrated in Figure (5.4.4) with $M = 24$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ and full precision arithmetic.

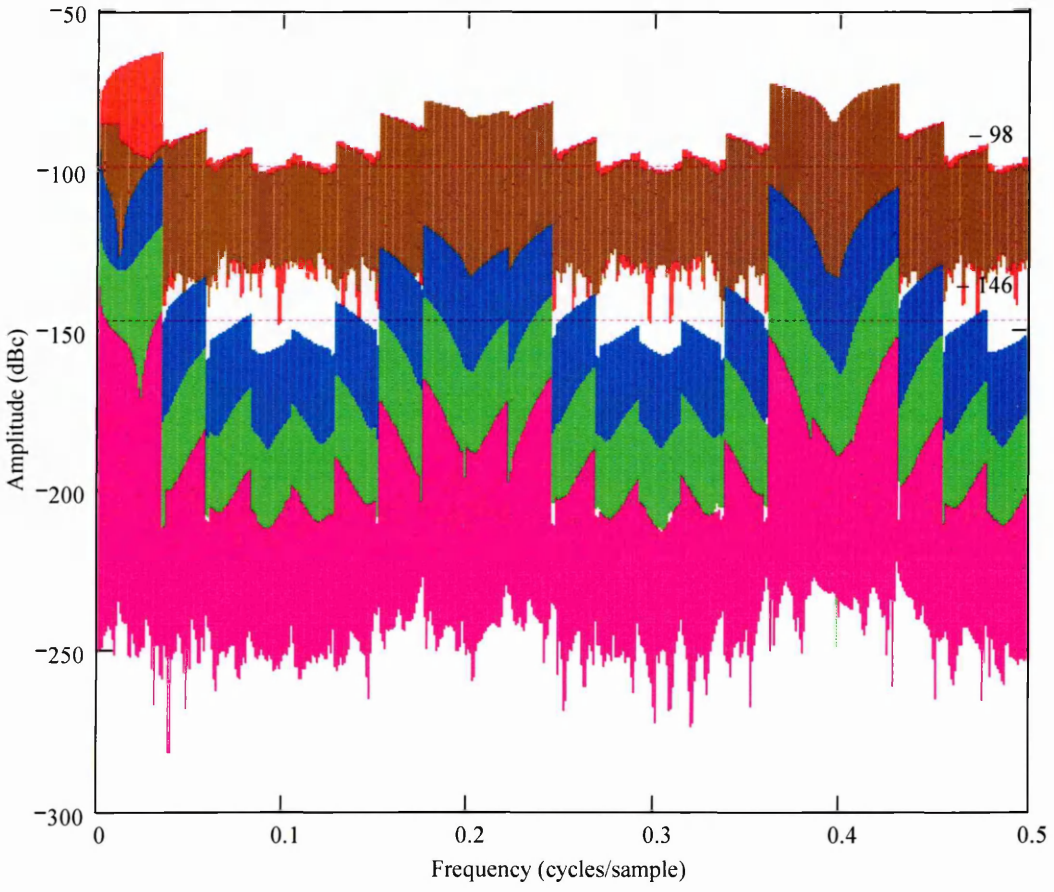


Figure (5.5.11): Composite amplitude error spectra for five interpolated addressing algorithms using a multi-harmonic wavetable tabulating a -3 dB/octave spectrum with $N_h = 100$ *as depicted in Figure (5.4.4).* $M = 24$, $I = 12$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*

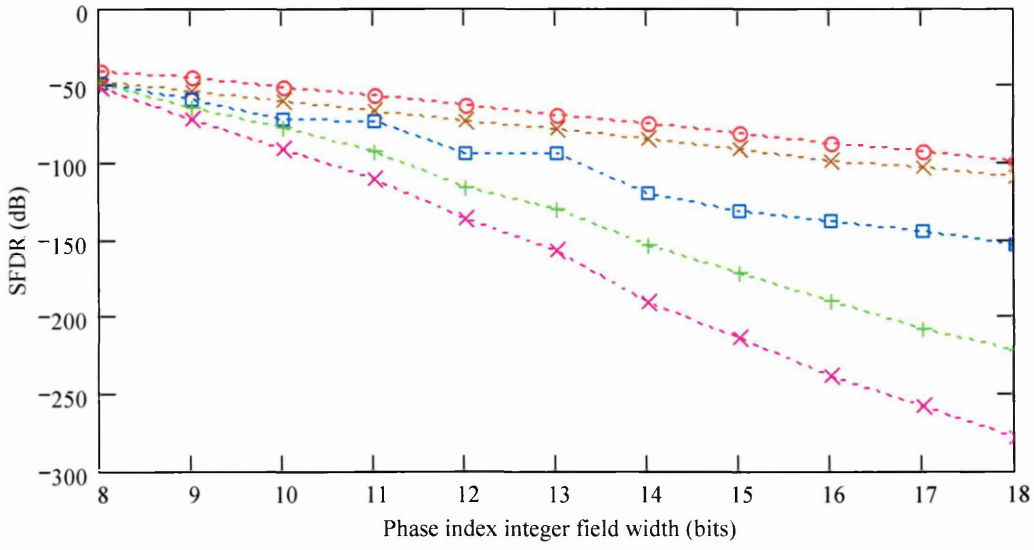
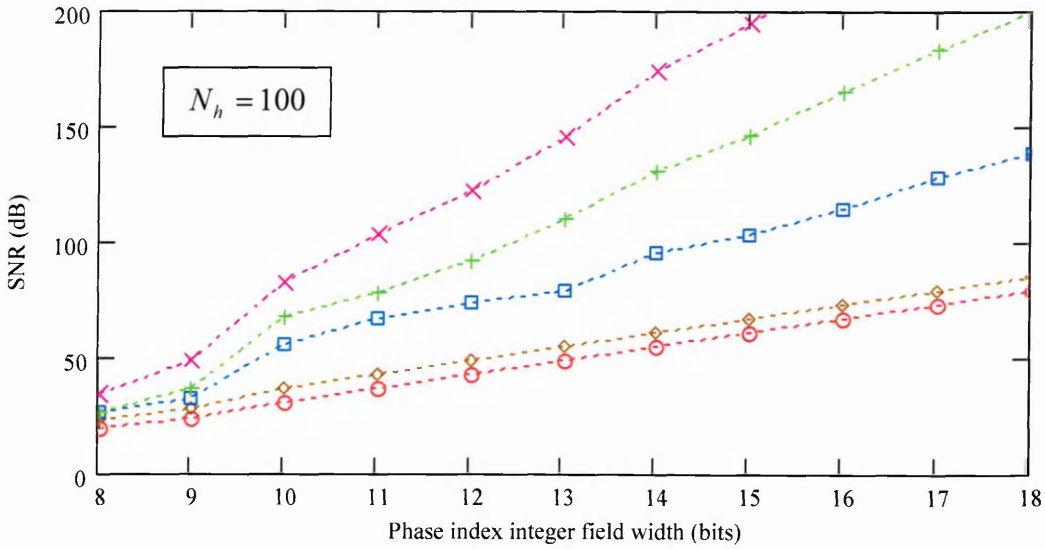


Figure (5.5.12): SFDR variation with $I \in [8,18]$ *and the phase mapping wavetable tabulating a multi-harmonic signal with -3 dB/octave spectrum as illustrated in Figure (5.4.4) with* $N_h = 100$, $M = 24$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*



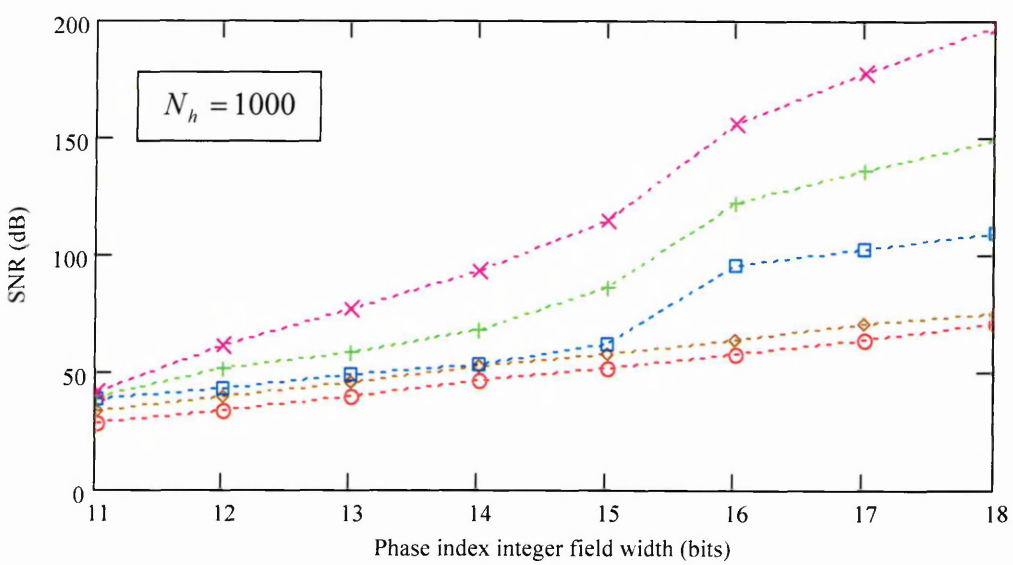


Figure (5.5.13): SNR variation with I for two N_h values. The phase mapping wavetable tabulates a multi-harmonic signal with -6 dB/octave spectrum as illustrated in Figure (5.4.4) with $M = 24$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ and full precision arithmetic.

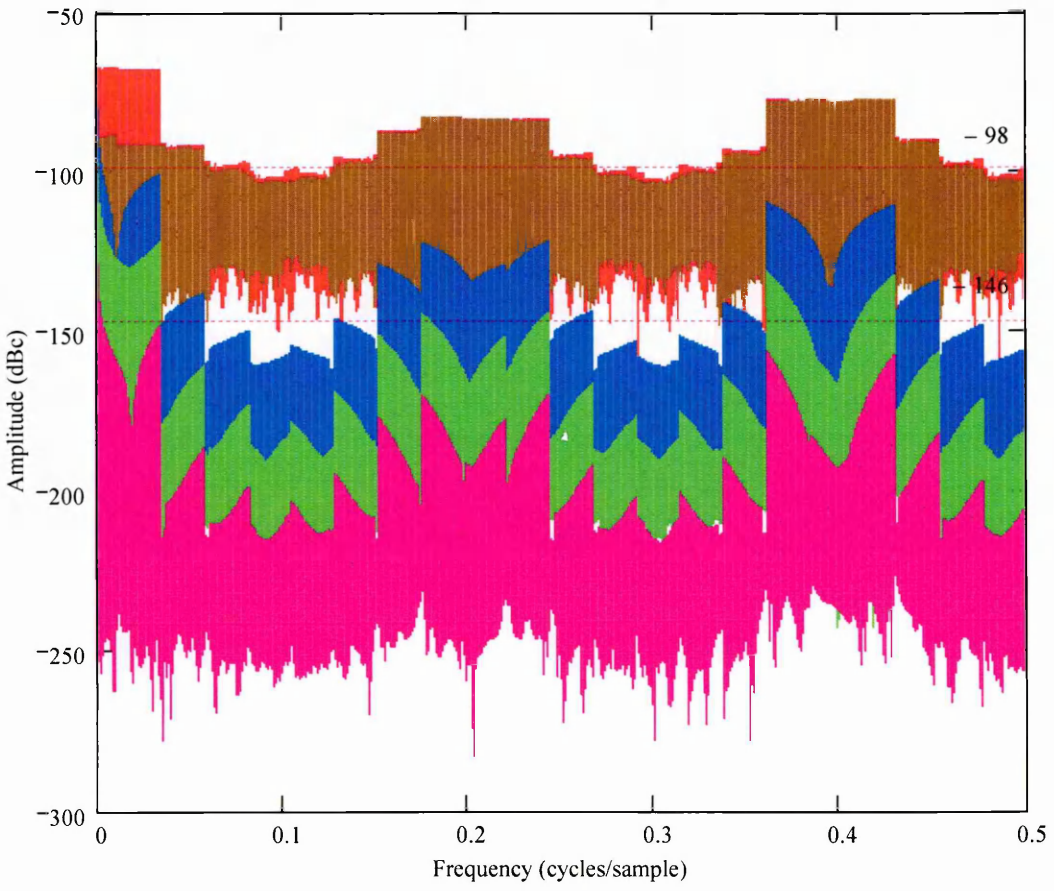


Figure (5.5.14): Composite amplitude error spectra for five interpolated addressing algorithms using a multi-harmonic wavetable tabulating a -6 dB/octave spectrum with $N_h = 100$ *as depicted in Figure (5.4.4).* $M = 24$, $I = 12$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*

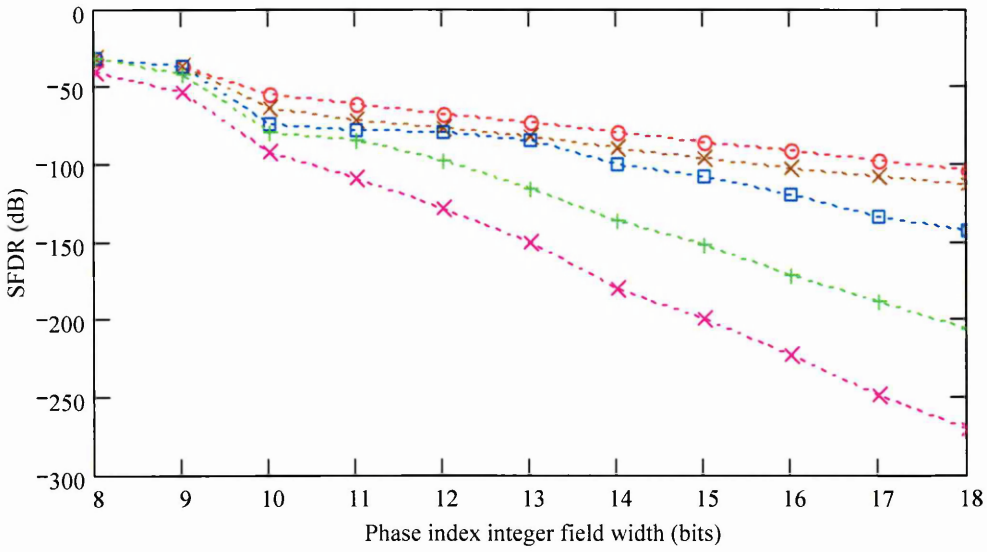
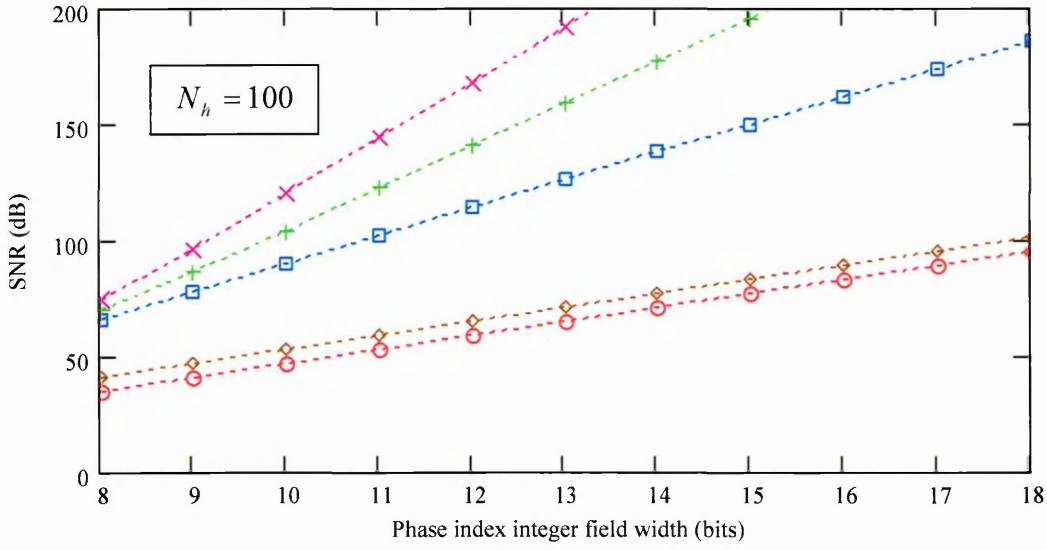


Figure (5.5.15): SFDR variation with $I \in [8,18]$ *and the phase mapping wavetable tabulating a multi-harmonic signal with -6 dB/octave spectrum as illustrated in Figure (5.4.4) with* $N_h = 100$, $M = 24$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*



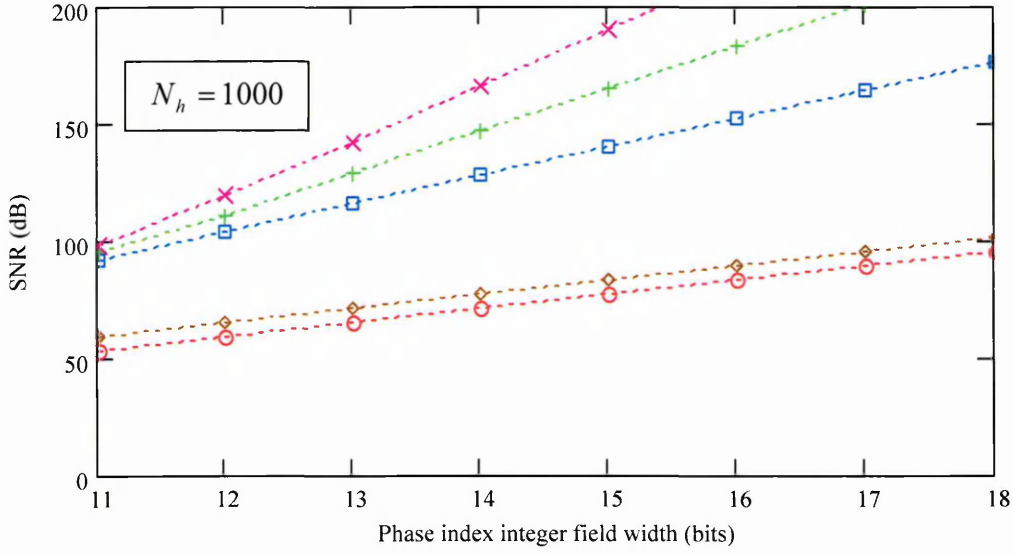


Figure (5.5.16): SNR variation with I for two N_h values. The phase mapping wavetable tabulates a multi-harmonic signal with -12 dB/octave spectrum as illustrated in Figure (5.4.4) with $M = 24$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ and full precision arithmetic.

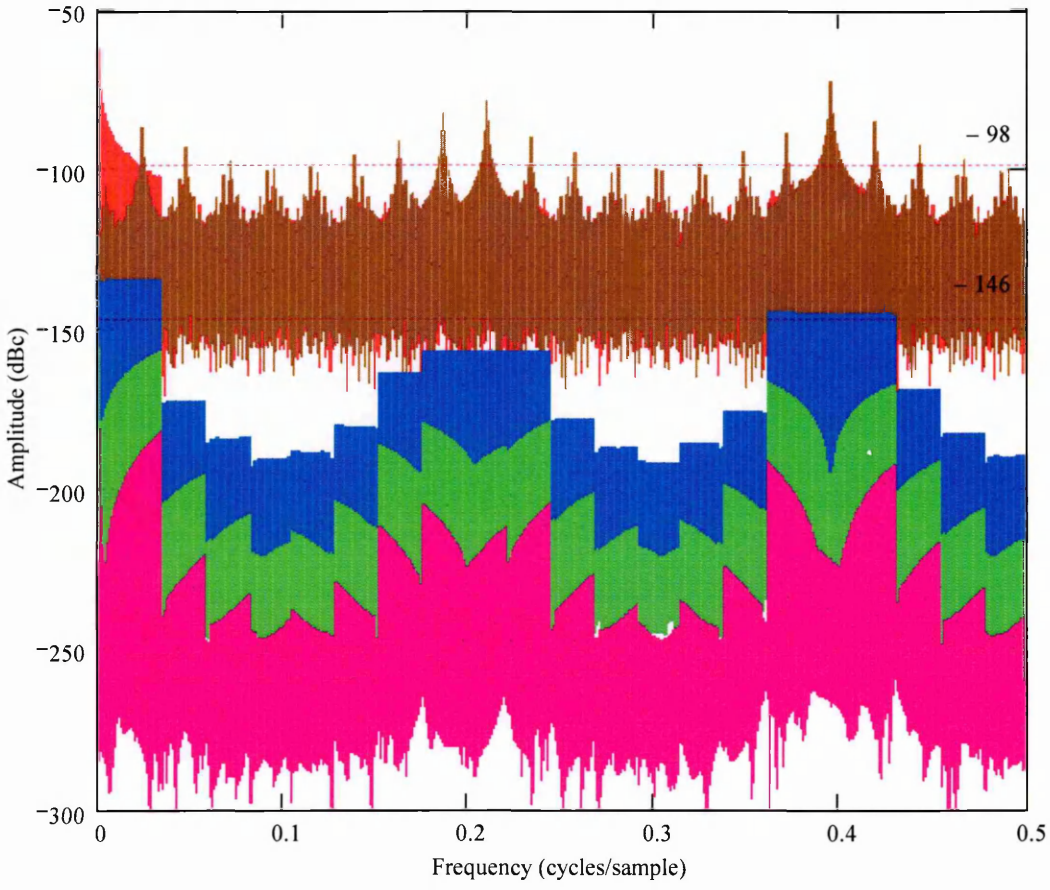


Figure (5.5.17): Composite amplitude error spectra for five interpolated addressing algorithms using a multi-harmonic wavetable tabulating a -12 dB/octave spectrum with $N_h = 100$ *as depicted in Figure (5.4.4).* $M = 24$, $I = 12$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*

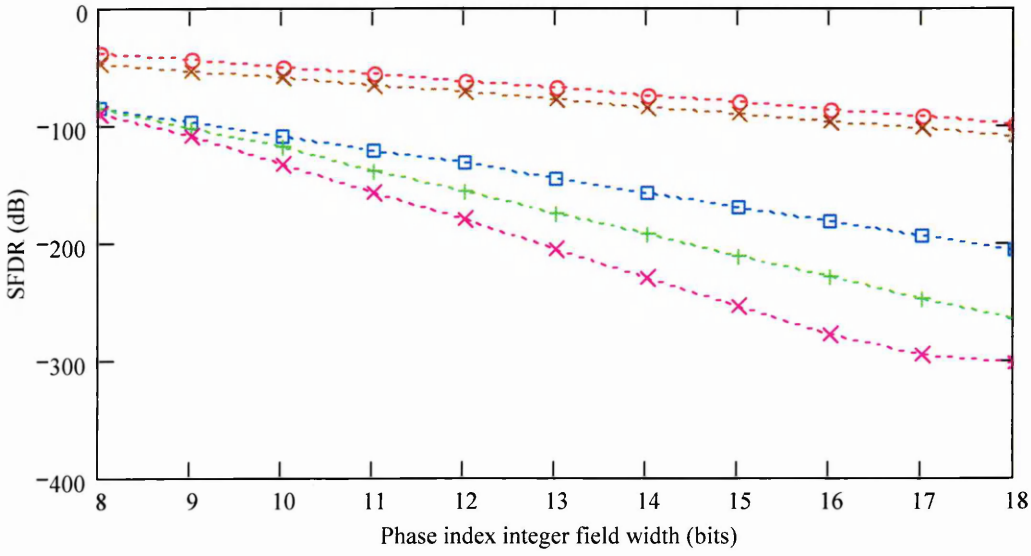
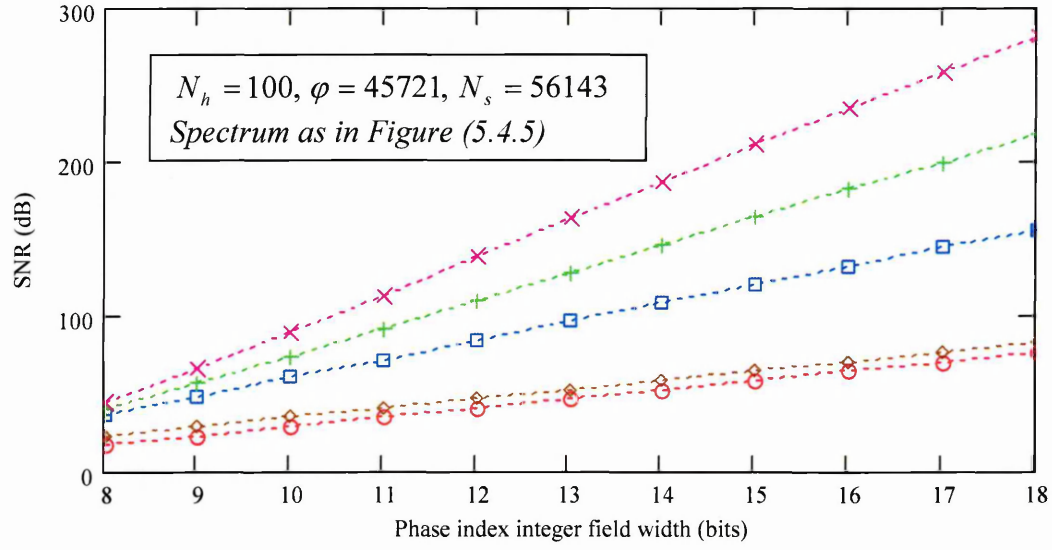


Figure (5.5.18): SFDR variation with $I \in [8,18]$. *The phase mapping wavetable tabulates a multi-harmonic signal with -12 dB/octave spectrum as illustrated in Figure (5.4.4) with* $N_h = 100$, $M = 24$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*



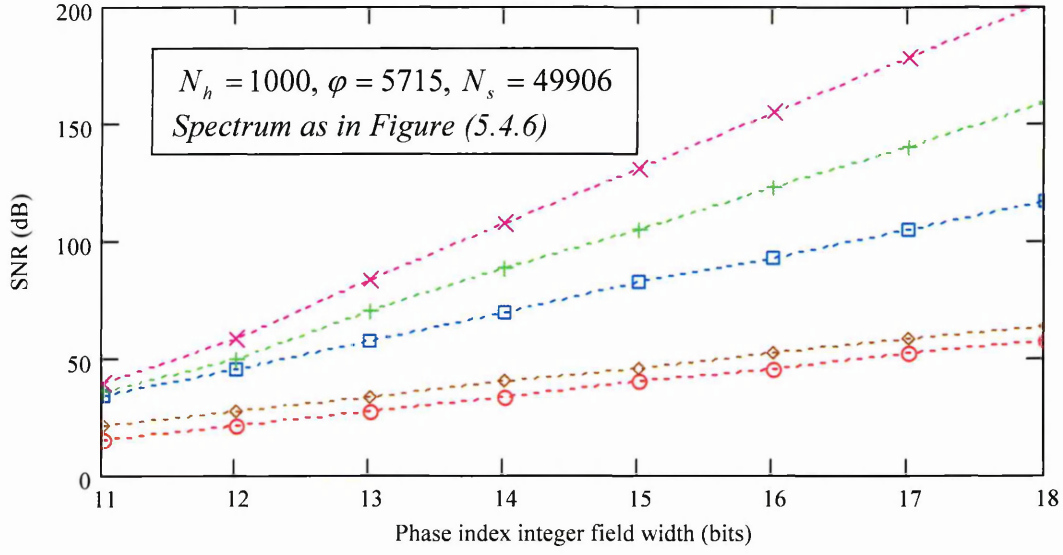


Figure (5.5.19): SNR variation with I for two N_h values using full precision arithmetic with $M = 24$ and $f_s = 48\text{KHz}$. The phase mapping wavetable tabulates a multi-harmonic signal with -12 dB/octave low-pass spectrum as illustrated in Figures (5.4.5) and (5.4.6).

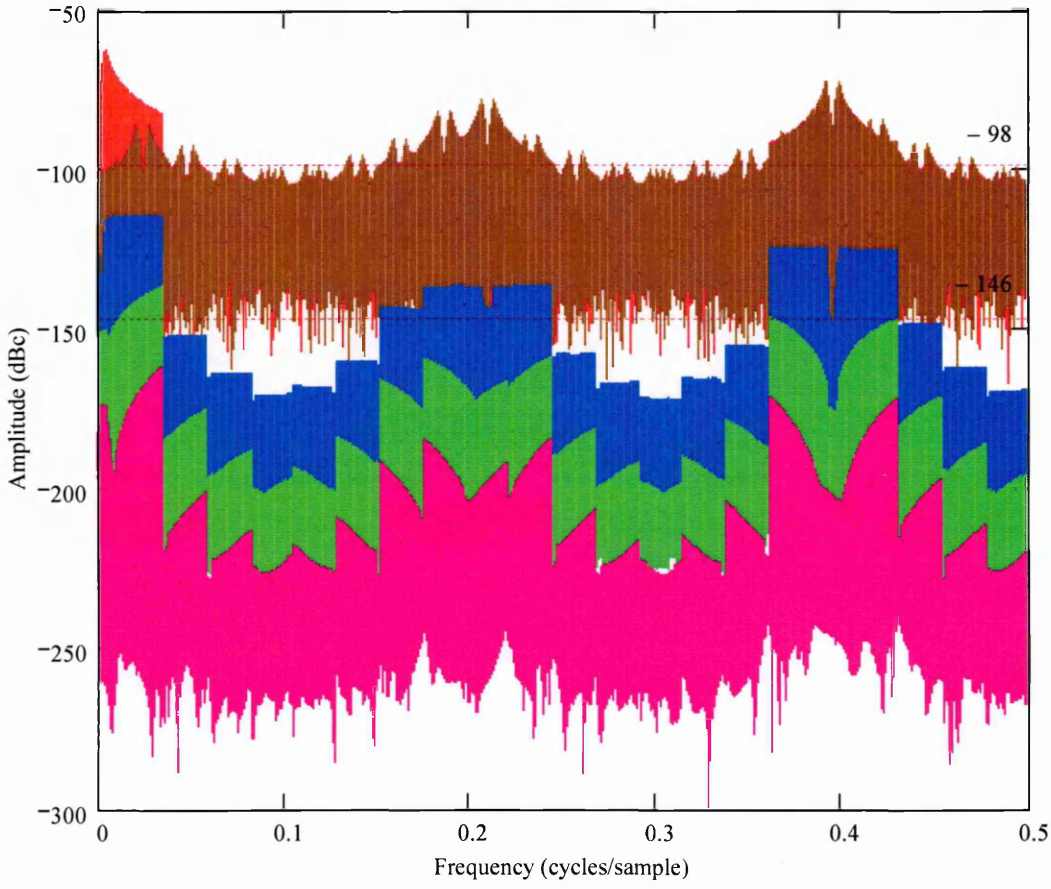


Figure (5.5.20): Composite amplitude error spectra for five interpolated addressing algorithms using a multi-harmonic wavetable tabulating a -12 dB/octave low-pass spectrum with $N_h = 100$ *as depicted in Figure (5.4.5).* $M = 24$, $I = 12$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*

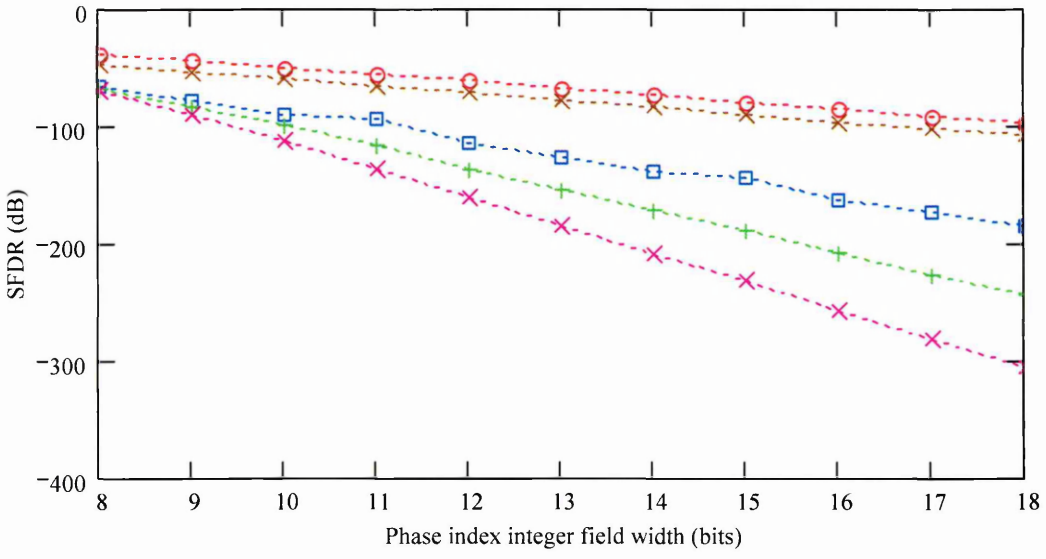
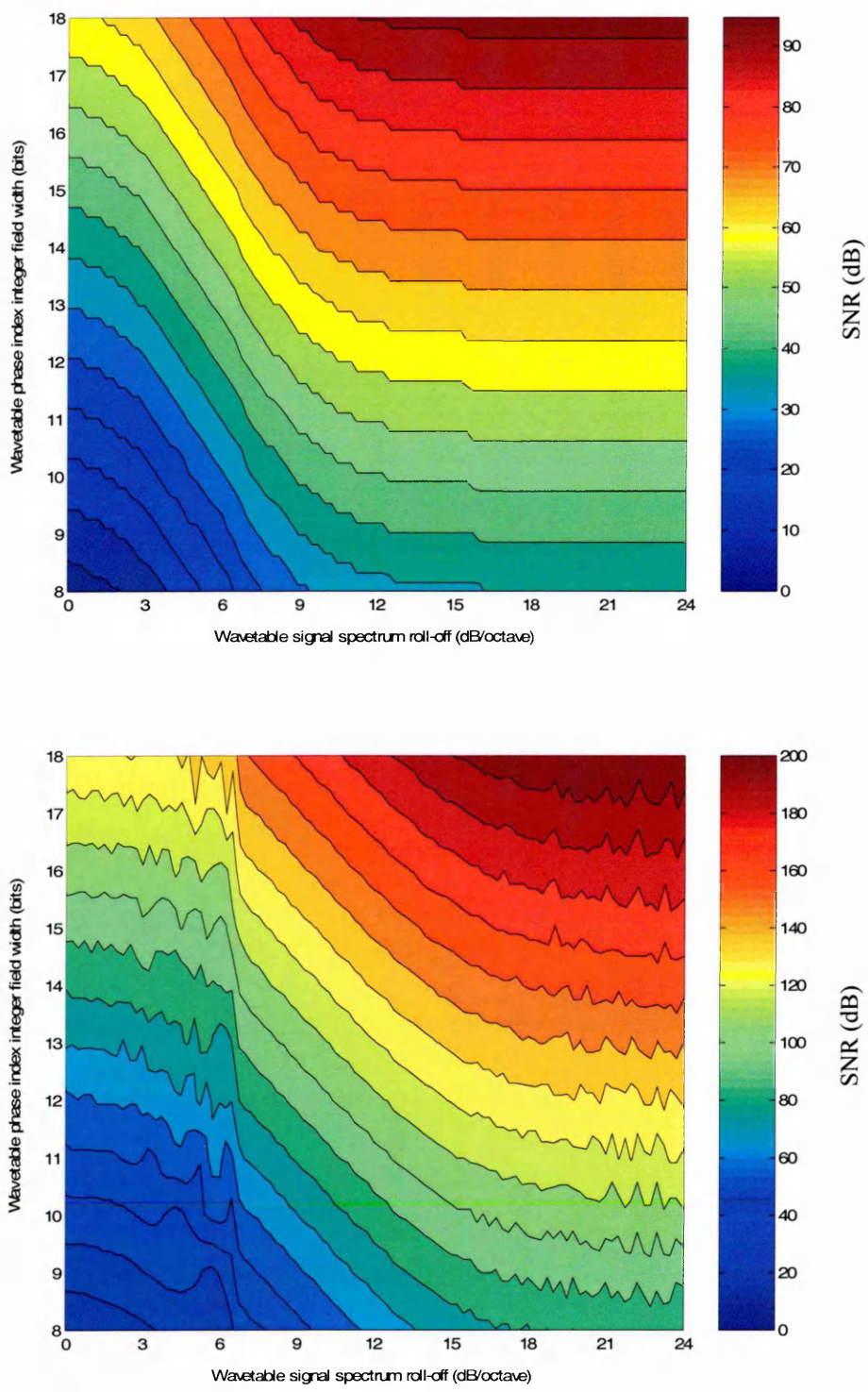
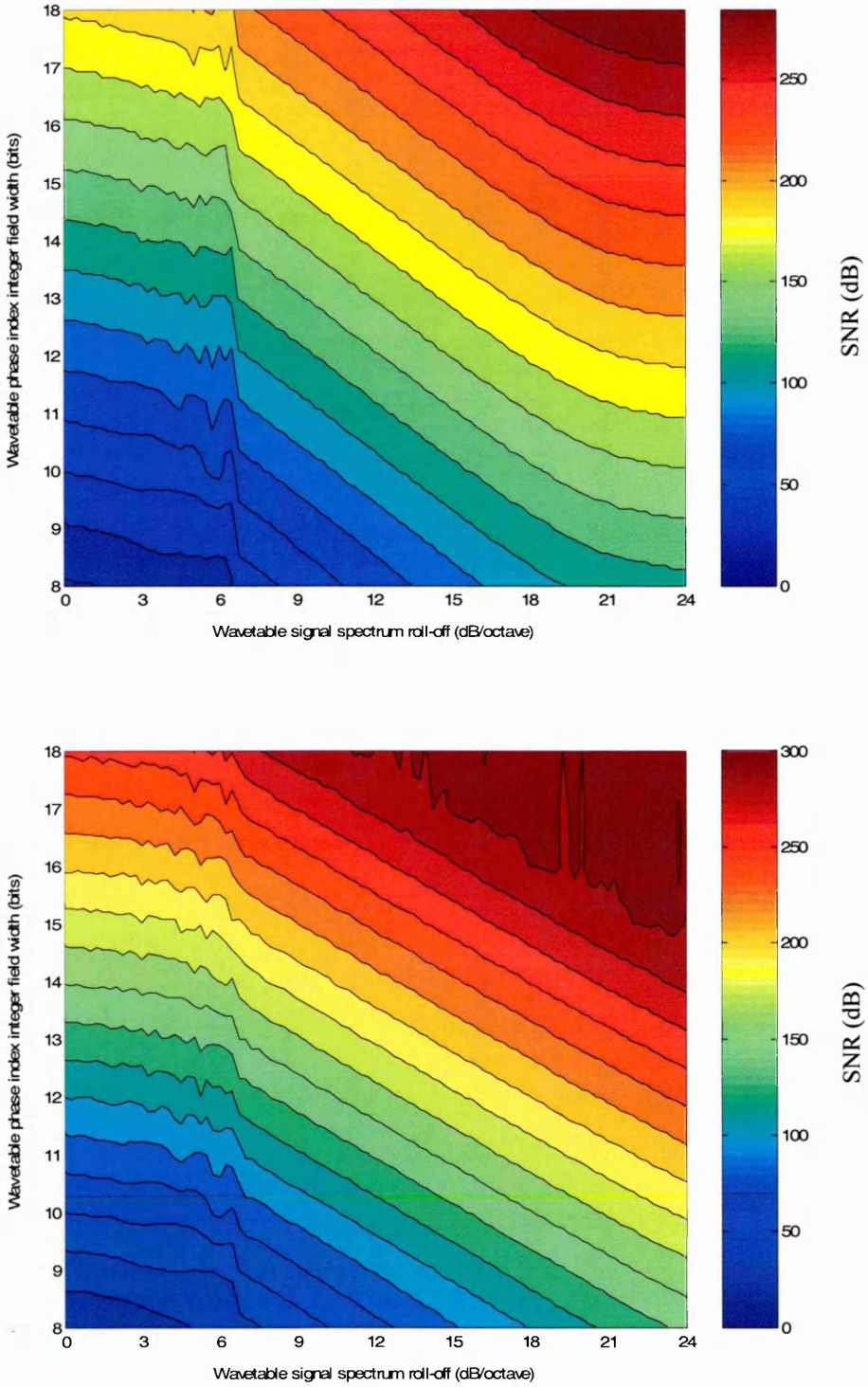


Figure (5.5.21): SFDR variation with $I \in [8,18]$ *and the phase mapping wavetable tabulating a multi-harmonic signal with -12 dB/octave low-pass spectrum as illustrated in Figure (5.4.5) with* $N_h = 100$, $M = 24$, $\varphi = 5715$, $N_s = 49906$, $f_s = 48\,\text{kHz}$ *and full precision arithmetic.*



Figures (5.5.22) and (5.5.23): SNR variation with I and wavetable spectrum roll-off slope ranging from 0 to -24 dB/octave over the first 100 harmonics using TPM (upper plot) and LIPM (lower plot). Contour lines illustrate loci of constant SNR.



Figures (5.5.24) and (5.5.25): SNR variation with I and wavetable spectrum roll-off slope ranging from 0 to -24 dB/octave over the first 100 harmonics using QIPM (upper plot) and CIPM (lower plot). Contour lines illustrate loci of constant SNR.

These results corroborate an intuitive view that the SNR of interpolated multi-harmonic WLS is a strong function of wavetable length, wavetable signal spectrum, interpolation order and arithmetic word size. We have simulated SNR using wavetable spectra which approximate the average spectral characteristic of typical musical signal classes discussed in section (5.4.4) and observe that for a given interpolation order and wavetable length, SNR *decreases* as the number and amplitude of tabulated harmonics *increases*. This result is not unexpected since higher harmonics are effectively stored in shorter wavetables compared to the fundamental. In general, SNR increases with increasing harmonic roll-off slope since upper harmonics become progressively less significant. For a given wavetable spectral characteristic, SNR increases with both interpolation order and wavetable length at an essentially constant rate. For an order N interpolation, we observe that SNR improves at approximately $6(N+1)$ dB/bit increment in I for all simulated spectra.

5.6 Conclusions

In this chapter we have investigated interpolated sinusoidal and multi-harmonic WLS and presented a sinusoidal phase mapping technique (TIPM) which represents an original contribution resulting from this research and is now published. Simulation results illustrate the qualitative performance of interpolated WLS against key parameter variations for five interpolation algorithms using the SNR metric under simulation conditions contrived to approximate typical and worst case music synthesis scenarios. Simulated amplitude error spectra illustrate the frequency domain distribution of noise components caused by interpolation errors and demonstrate the effectiveness of increasing interpolation order in reducing the amplitude of these components.

It is clear from simulation results summarised in Table (5.6.1), that LIPM and TIPM are preferred for sinusoidal synthesis, giving SNR performance comparable to 24-bit SQNR with wavetable lengths of 8192 samples. Indeed, we confirm the hypothesis that sample quantisation noise defines the SNR bound for TIPM. Attaining comparable SNR performance between TPM and LIPM requires a TPM wavetable around 256 times larger than the LIPM wavetable and is likely to be impractical in most cases. We conclude that the QIPM and CIPM algorithms are unsuitable for sinusoidal synthesis since they impose an excessive computational burden compared to LIPM and TIPM which yield acceptable SNR performance with reasonable wavetable lengths. TIPM reduces phase truncation errors to the quantisation noise floor, but imposes greater arithmetic and table lookup overhead compared to LIPM. TIPM memory overhead reduces exponentially with M and so becomes favourable for smaller phase accumulator word lengths and hence reduced frequency resolution. TIPM finds utility in applications requiring quadrature sinusoids with optimal SNR and phase control precision bound by M alone.

Property	TPM	RPM	LIPM	QIPM	CIPM	TIPM *Note 1* $M=24$	TIPM *Note 1* $M=20$
Minimum multiply operations (incl. $A(n)$)	1(2) *Note 2*	1(2) *Note 2*	2(4) *Note 2*	4(8) *Note 2*	5(10) *Note 2*	3(4) *Note 2*	3(4) *Note 2*
Add/subtract operations	0	1 *Note 3*	1	2	3	1 *Note 2*	1 *Note 2*
Wavetable lookups	1	1	2 *Note 4*	3 *Note 4*	4 *Note 4*	4 *Note 4*	4 *Note 4*
Coefficient table lookups *Note 5*	0	0	0	3	4	0	0
Total table lookups	1	1	2	6	8	4	4
Total coefficient lookup table size *Note 5*	0	0	0	$3(2^F)$	$4(2^F)$	0	0
Wavetable size for SNR ≥ 16-bit SQNR *Note 6*	2^{18}	2^{17}	$2(2^9)$ 1024	$3(2^7)$ 384	$4(2^6)$ 256	$4(2^{12})$ *Note 7* 16384	$4(2^{10})$ *Note 7* 4096
TIPM wavetable size with $R>0$ *Note 7* and SNR ≥ 16-bit SQNR	-	-	-	-	-	$4(2^9)$ 2048 $R=6$	$4(2^8)$ 1024 $R=2$
Wavetable size for SNR ≥ 24-bit SQNR *Note 6*	$>>2^{18}$	$>>2^{18}$	$2(2^{13})$ 16384	$3(2^{10})$ 3072	$4(2^8)$ 1024	$4(2^{12})$ *Note 7* 16384	$4(2^{10})$ *Note 7* 4096

1 TIPM is *only* applicable to sinusoidal phase mapping.
2 Bracketed value indicates arithmetic overhead for quadrature sinusoid phase mapping.
3 Rounding the truncated phase word requires a conditional addition operation.
4 k table lookups require k accesses of a single memory or k separate memories.
5 Interpolation coefficients computed by lookup tables indexed with phase fraction.
6 Total wavetable size assumes separate memory for each interpolation term.
7 TIPM memory utilisation is optimal when $L=2^{\frac{M}{2}}$ with SNR always bound by SQNR.

Table (5.6.1): Summary arithmetic overhead and SNR performance for our six interpolation algorithms applied to sinusoidal phase-amplitude mapping.

Table (5.6.2) summarises simulated SNR performance for our five interpolation algorithms using wavetables containing signals with multi-harmonic spectra. We conclude that wavetable lengths consistent with a 16-bit SQNR using TPM and RPM are impractically large for all wavetable test spectra simulated here. Wavetable lengths consistent with a 24-bit SQNR are higher still and hence we exclude TPM and RPM on cost-effectiveness grounds. For our range of wavetable test spectra, QIPM requires wavetable lengths (L) between 65,536 and 1024 samples for SNR comparable with a 16-bit SQNR and between 524,288 and 8192 samples for SNR comparable with a 24-bit SQNR. Additionally, CIPM requires wavetable lengths between 32,768 and 512 samples for SNR comparable with a 16-bit SQNR and between 131,072 and 2048 samples for SNR comparable with a 24-bit SQNR.

We conclude that CIPM provides optimum SNR performance given our range of wavetable spectra, arithmetic quantisation and interpolation orders. However, the simulation models presented in Appendix B are readily configurable to simulate higher order interpolation algorithms if required. CIPM requires four wavetable read operations and four coefficient multiplication operations, assuming interpolation coefficients are readily available for a given $\phi_F(n)$ value. When selecting a wavetable length for a particular worst case SNR, we must also consider the inverse relationship between wavetable length and "fill time" (i.e. the time to move new samples from a mass storage device into the wavetable). In Chapter 6 we present an original wavetable memory architecture that provides a data-parallel, length-N sample set supporting an order-N interpolation without imposing $(N+1)$ fold memory redundancy.

Wavetable Spectrum	Simulation Condition	Wavetable Length, L			
		TPM *Note 1*	LIPM	QIPM	CIPM
-3 dB linear *Note 2*	SNR ≥ 16-bit SQNR	$>2^{18}$	32768	8192	4096
$N_h = 100$	SNR ≥ 24-bit SQNR	$>>2^{18}$	$>2^{18}$	32768	16384
-3 dB linear *Note 2*	SNR ≥ 16-bit SQNR	$>>2^{18}$	2^{18}	65536	32768
$N_h = 1000$	SNR ≥ 24-bit SQNR	$>>2^{18}$	$>>2^{18}$	2^{19}	131072
-6 dB linear *Note 2*	SNR ≥ 16-bit SQNR	$>2^{18}$	32768	8192	2048
$N_h = 100$	SNR ≥ 24-bit SQNR	$>>2^{18}$	$>2^{18}$	32768	8192
-6 dB linear *Note 2*	SNR ≥ 16-bit SQNR	$>2^{18}$	2^{18}	65536	32768
$N_h = 1000$	SNR ≥ 24-bit SQNR	$>>2^{18}$	$>>2^{18}$	2^{18}	65536
-12 dB linear *Note 2*	SNR ≥ 16-bit SQNR	2^{18}	2048	1024	512
$N_h = 100$	SNR ≥ 24-bit SQNR	$>>2^{18}$	32768	8192	2048
-12 dB linear *Note 2*	SNR ≥ 16-bit SQNR	2^{18}	4096	2048	2048
$N_h = 1000$	SNR ≥ 24-bit SQNR	$>>2^{18}$	65536	16384	16384
-12 dB low-pass	SNR ≥ 16-bit SQNR	2^{18}	8192	4096	2048
$N_h = 100$ *Note 3*	SNR ≥ 24-bit SQNR	$>>2^{18}$	131072	16384	8192
-12 dB low-pass	SNR ≥ 16-bit SQNR	2^{18}	131072	32768	16384
$N_h = 1000$ *Note 4*	SNR ≥ 24-bit SQNR	$>>2^{18}$	$>2^{18}$	2^{18}	65536

1 RPM wavetable length results are not summarised since they are consistently around one half of the TPM value for all simulation conditions.
2 Linear roll-off spectral response as illustrated in Figure (5.4.4).
3 PWL low-pass spectral response as illustrated in Figure (5.4.5).
4 PWL low-pass spectral response as illustrated in Figure (5.4.6).

Table (5.6.2): Summary characteristics of four interpolation algorithms applied to multi-harmonic phase-amplitude mapping.

Chapter 6 Arithmetic Processing Architectures

6.1 Introduction

In this Chapter we investigate arithmetic processing architectures which implement two key synthesis techniques presented earlier in this thesis – *fractional wavetable addressing* investigated in Chapter 5 and *phase domain processing* introduced in Chapter 4, which we define as *the algorithmic processing of a DT phase sequence prior to phase-amplitude mapping to effect frequency or phase control of a synthesised partial.* Phase domain processing underpins an entire synthesis subclass that efficiently implements the HAS and PAS processing models introduced in Chapter 2.

The importance of fractional memory addressing within the WLS algorithm motivates investigation of wavetable memory architectures which are compatible with the interpolation processing models which effect computation of the fractional value. A fundamental premise of the fractional memory addressing model presented in Chapter 5 is a "manifold sample set" *centred* about the sample addressed by the integer phase index component. This sample set together with coefficients formed from the phase fraction component, are used to compute the interpolated output sample which represents an approximation to the fractionally addressed sample within an error bound governed by the interpolation order. An order-N interpolation requires $(N+1)$ memory read operations causing a processing bottleneck in time-multiplexed implementations using multiple access of a single memory. We therefore require a wavetable memory architecture which generates manifold sample sets with a *single* parallel read operation maximising throughput through data-parallel interpolation processing. This is discussed in section (6.2).

In section (6.3) we develop the concept of *phase domain processing* and present new sequential processing models that result directly from this research which synthesise manifold, independently controlled partials based on the work of Chamberlin [1976]. These architectures are readily pipelined to provide computational throughput in line with the requirements of a multi-voice implementation of the generalised PAS model.

6.2 Memory Access and Interpolated Fractional Addressing

6.2.1 Consecutive Access Vector Memory

In this section we present an extensible, multi-port memory architecture that enables simultaneous access to a *consecutive block* or *vector* of data from a single logical base address and represents original work from this research. This architecture is motivated primarily by the data requirements of interpolated WLS discussed in Chapter 5, which requires $(N+1)$ wavetable samples (i.e. memory read operations) for an order-N interpolation computation. Fast execution of this algorithm using data-parallel arithmetic processing therefore requires that all $(N+1)$ wavetable samples are available simultaneously from a single read operation of a *parallel* structured wavetable memory. In the following discussion, we drop sequence time-indices for brevity and use ϕ_I and α to represent the DT sequences $\phi_I(n)$ and $\alpha(n)$, respectively. For a logical address, ϕ_I, indexing an arbitrary wavetable, which we denote by the vector $\mathbf{T}$, this multi-port memory architecture produces the set of k consecutive wavetable samples $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I+1], \mathbf{T}[\phi_I+2],\ldots, \mathbf{T}[\phi_I+(k-1)]\}$ from k distinct output ports. We term this memory architecture an order-k *consecutive access vector memory* (CAVM) which arranges the *physical* memory into k independently addressable memory blocks with

respective data output ports, each providing a particular data element from the sample set[1].

The essence of the extensible CAVM technique is to arrange sample storage so that *any* k consecutive samples are stored unambiguously across k memory blocks. However, the CAVM addressing strategy causes the *order* of the k samples to take on k cyclic permutations of the consecutively ordered wavetable sample set $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I+1], \mathbf{T}[\phi_I+2],\ldots, \mathbf{T}[\phi_I+(k-1)]\}$ with respect to the k output ports depending on the address ϕ_I. In addition to memory configuration, two other functions underpin the CAVM architecture – logical to physical address translation and output data reordering.

Optimal order-N interpolation (i.e. providing minimum interpolation error bound) requires the fractional address to lie within the middle sub-interval of the sample set for odd-order interpolations or within either sub-interval about the middle sample of the sample set for even order interpolation. By adding an offset, j, to the logical CAVM address, the sample set is positioned arbitrarily with respect to $\mathbf{T}[\phi_I]$ and we obtain the sample set $\{\mathbf{T}[\phi_I+j], \mathbf{T}[\phi_I+j+1], \mathbf{T}[\phi_I+j+2],\ldots, \mathbf{T}[\phi_I+j+(k-1)]\}$. By setting $j=-\left\lfloor\frac{N-1}{2}\right\rfloor$ when $k=N+1$, we obtain the sample set required for an order-N interpolation defined by Eq. (5.2.2) yielding minimum interpolation error bound. In the following discussion we describe CAVM operation using a non offset logical base address to simplify notation and presentation of the concept.

[1] For our present discussion we are not concerned with how the CAVM is written with wavetable data. In a physical realisation of an order-k CAVM, a dedicated multiplexer associated with each memory address and data port provides a dedicated direct memory access (DMA) port for memory write operations.

6.2.2 The Order-2 CAVM and Linear Interpolation

We demonstrate the CAVM principle by considering the simplest order-2 architecture that produces the sample set $\{\mathbf{T}[\phi_l], \mathbf{T}[\phi_l + 1]\}$ as required for a linear interpolation of a wavetable vector $\mathbf{T}$ (i.e. $N = 1$ and $k = 2$). We assume a memory vector $\mathbf{T}$, of length L', which is exactly divisible by 2 that contains an *integer number* of single cycle wavetables each of length L that we index with the parameter ψ (not to be confused with the phase function denotation of section (4.2.3)). The order-2 CAVM comprises two distinct memory blocks denoted by the vectors $\mathbf{B}_0$ and $\mathbf{B}_1$, each of length $\frac{L'}{2}$ samples where block $\mathbf{B}_0$ holds *even* address samples and block $\mathbf{B}_1$ holds *odd* address samples of the vector $\mathbf{T}$. Blocks $\mathbf{B}_0$ and $\mathbf{B}_1$ comprise a vector of k-spaced wavetable data samples as exemplified in Figure (6.2.1) for the illustrative case with $L = 8$.

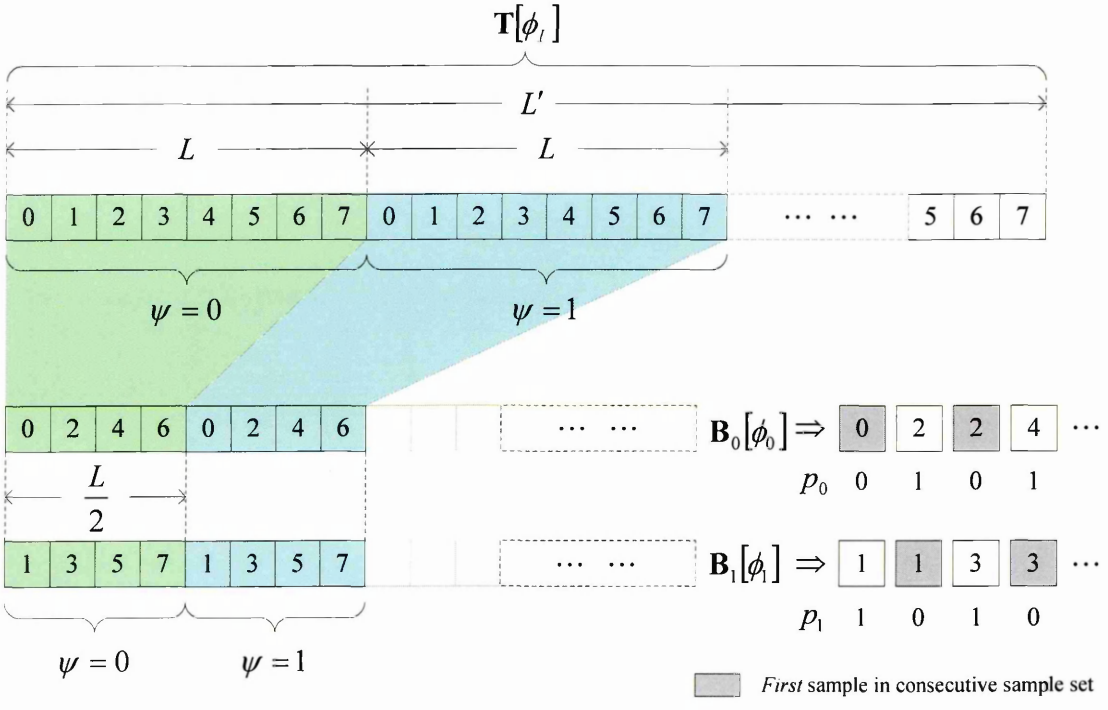


Figure (6.2.1): Memory allocation for the order-2 CAVM with $L = 8$.

It is clear that memory blocks $\mathbf{B}_0$ and $\mathbf{B}_1$ are written with data from $\mathbf{T}$ according to:

$$\mathbf{B}_0[m] = \mathbf{T}[n] \quad \text{for} \quad n = 2m$$

$$\mathbf{B}_1[m] = \mathbf{T}[n] \quad \text{for} \quad n = 2m+1 \qquad (6.2.1)$$

$$m \in [0, \frac{L'}{2} - 1]$$

Assuming radix-2 L and L' values, we partition the $\log_2(L')$-bit logical address into two components: a *phase* index, $\phi_I \in [0, L-1]$, comprising the $\log_2(L)$ least significant bits that address a particular wavetable *sample* modulo-L and a *wavetable* index component, $\psi \in [0, \frac{L'}{L} - 1]$, comprising the $\log_2(L') - \log_2(L)$ most significant bits that address a particular *wavetable* from the set of $\frac{L'}{L}$ wavetables. In terms of an abstract memory model, the wavetable index can be considered as an integer "page address", with each page storing an individual wavetable. Hypothetically, we may extend this concept to define a *fractional* wavetable address (i.e. a fractional ψ parameter) which we interpret as an *interpolation between consecutive wavetables* in the set. This is discussed in section (6.2.5).

The phase index, ϕ_I, is transformed into two physical block addresses, $\phi_0 \in [0, \frac{L}{2} - 1]$ and $\phi_1 \in [0, \frac{L}{2} - 1]$[2] which respectively address memory blocks $\mathbf{B}_0$ and $\mathbf{B}_1$ modulo-$\frac{L}{2}$. The ϕ_0 and ϕ_1 block addresses are defined so as to index the $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1]\}$ sample set as ϕ_I varies over the range of permissible values (i.e. $\phi_I \in [0, 2^I - 1]$). For the order-2 CAVM and arbitrary L, the block addresses are given by:

[2] The reader should observe the distinction between the first block address, ϕ_1, and the truncated phase component ϕ_I to avoid confusion.

$$\phi_0 = \left\langle \left\lfloor \frac{\phi_I + 1}{2} \right\rfloor \right\rangle_{\frac{L}{2}} \qquad \phi_1 = \left\langle \left\lfloor \frac{\phi_I}{2} \right\rfloor \right\rangle_{\frac{L}{2}} \tag{6.2.2}$$

Additionally, we define *sample-type indices*, p_0 and p_1, each associated with a respective memory block output that indicate the *position* of a corresponding data value within the ordered sample set (i.e. the first or second element in the order-2 CAVM sample set). Table (6.2.1) illustrates the block address and sample-type index sequences for an order-2 CAVM with $L = 8$.

ϕ_I	ϕ_0	ϕ_1 [3]	$\mathbf{B}_0[\phi_0]$	p_0	$\mathbf{B}_1[\phi_1]$	p_1	**Sample Set**
0	0	0	**T**[0]	0	**T**[1]	1	{**T**[0], **T**[1]}
1	1	0	**T**[2]	1	**T**[1]	0	{**T**[1], **T**[2]}
2	1	1	**T**[2]	0	**T**[3]	1	{**T**[2], **T**[3]}
3	2	1	**T**[4]	1	**T**[3]	0	{**T**[3], **T**[4]}
4	2	2	**T**[4]	0	**T**[5]	1	{**T**[4], **T**[5]}
5	3	2	**T**[6]	1	**T**[5]	0	{**T**[5], **T**[6]}
6	3	3	**T**[6]	0	**T**[7]	1	{**T**[6], **T**[7]}
7	0	3	**T**[0]	1	**T**[7]	0	{**T**[7], **T**[0]}

Table (6.2.1): Order-2 CAVM address sequences, memory block data values and sample-type indices for $L = 8$. The $\mathbf{B}_0[\phi_0]$ and $\mathbf{B}_1[\phi_1]$ columns illustrate sample ordering permutations at the memory block outputs.

[3] The reader should observe the distinction between the first block address, ϕ_1, and the truncated phase integer component ϕ_I to avoid confusion.

The logical address transformation illustrated in Table (6.2.1) causes the memory blocks to output the sample set $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1]\}$, but with a sample *order* dependent on the value of ϕ_I as reflected in the values of p_0 and p_1. If ϕ_I is even or zero, $\mathbf{T}[\phi_I]$ sample-types are output from block $\mathbf{B}_0$ and $\mathbf{T}[\phi_I + 1]$ sample-types are output from block $\mathbf{B}_1$. Conversely, if ϕ_I is odd, $\mathbf{T}[\phi_I + 1]$ sample-types are output from block $\mathbf{B}_0$ and $\mathbf{T}[\phi_I]$ sample-types are output from block $\mathbf{B}_1$. For our order-2 CAVM example we are therefore concerned with determining whether ϕ_I is odd or even and so p_0 is the least significant bit of the phase index, ϕ_I. Conversely, p_1 is the *inverted* least significant bit of the phase index, ϕ_I.

In general, the quotient $\frac{L}{k}$ must take on exact *integer* values to ensure that consecutive samples are unambiguously allocated across memory blocks. For example, with $L = 7$ and hence $\frac{L}{2} \notin Z$, we observe from Figure (6.2.1) that $\mathbf{T}[0]$ and $\mathbf{T}[6]$, which are consecutive modulo-7, are both stored in memory block $\mathbf{B}_0$ and hence cannot be accessed *simultaneously* and so preventing data parallelism within subsequent interpolation processing. In general, constraining L', L and k to take on only radix-2 values guarantees unique allocation of waveform samples across memory blocks with modulo-$\frac{L}{k}$ addressing wraparound. Figure (6.2.2) illustrates the arithmetic processing model for an order-2 CAVM and linear interpolation processor assuming a fractional phase address, $(\phi_I + \alpha)$ and wavetable index, ψ.

Two 2-to-1 multiplexers controlled by p_0 and p_1 reorder the memory block outputs into a contiguous sequence of ordered sample pairs, $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1]\}$, which feed the first-order difference subtractor. The first-order difference is multiplied by α and then

added to $\mathbf{T}[\phi_I]$ to produce the interpolated sample $y(n)$. Alternatively, one of the multiplexers becomes superfluous if the output adder is modified to accept a control input which *reverses* the sign of the input fed from the multiplier according to the sense of p_0 or p_1 as illustrated in Figure (6.2.3).

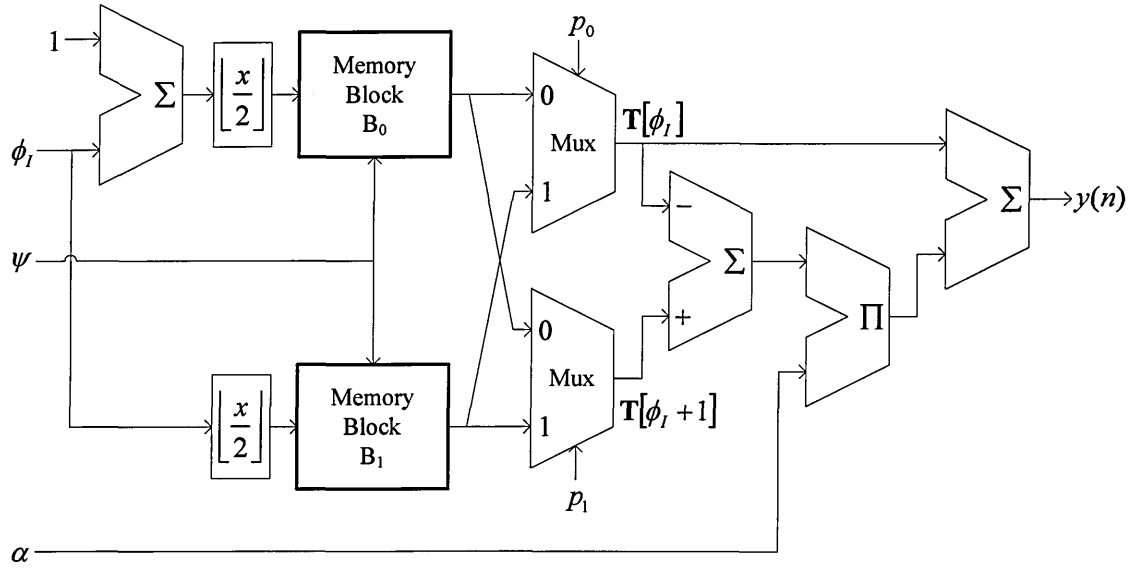


Figure (6.2.2): Order-2 CAVM and linear interpolation processing model.

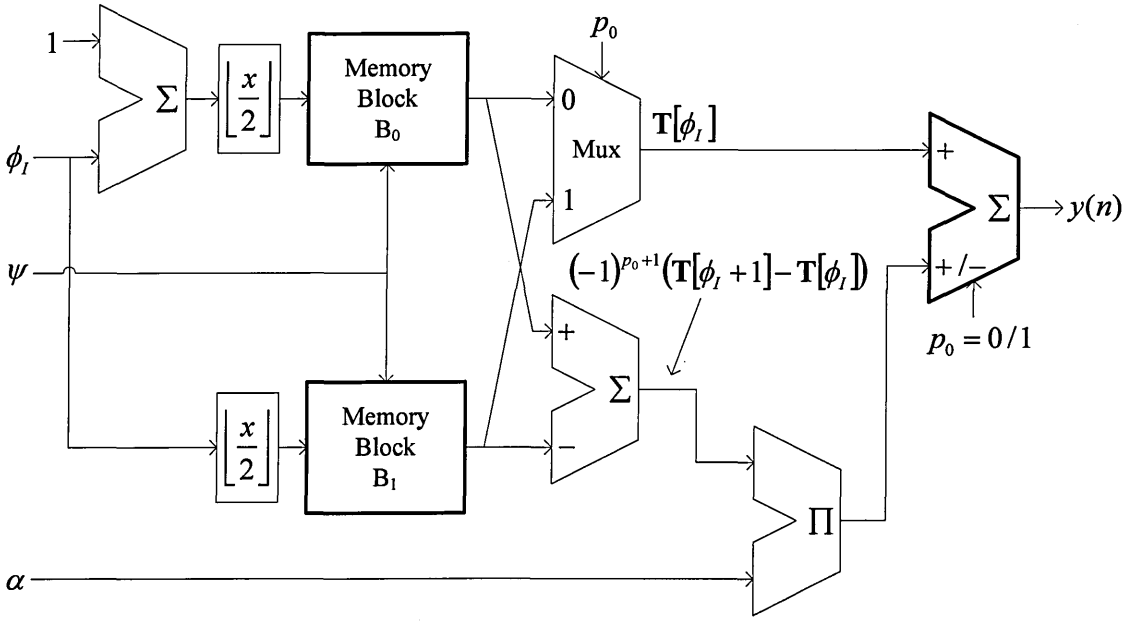


Figure (6.2.3): Order-2 CAVM and linear interpolation processing with reduced multiplexer count.

6.2.3 The Order-4 CAVM and Cubic Interpolation

The order-4 CAVM architecture comprises four distinct memory blocks, $\mathbf{B}_0$, $\mathbf{B}_1$, $\mathbf{B}_2$ and $\mathbf{B}_3$, generating the data-parallel sample set $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1], \mathbf{T}[\phi_I + 2], \mathbf{T}[\phi_I + 3]\}$ needed for a cubic interpolation of the wavetable vector $\mathbf{T}$. Samples are allocated to individual memory blocks from $\mathbf{T}$ in increments of four as illustrated in Figure (6.2.4) for our $L = 8$ example.

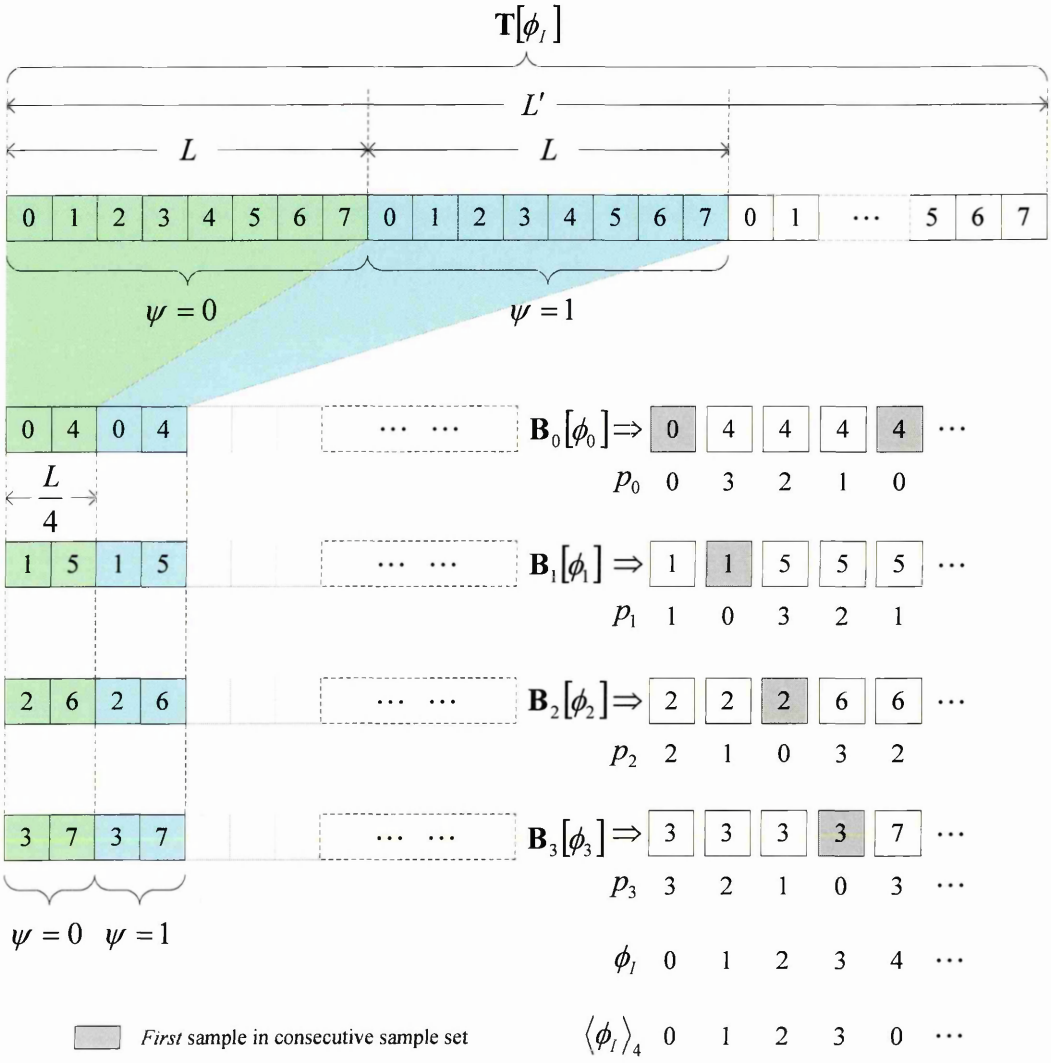


Figure (6.2.4): Memory allocation for the order-4 CAVM with $L = 8$.

Memory blocks $\mathbf{B}_0$, $\mathbf{B}_1$, $\mathbf{B}_2$ and $\mathbf{B}_3$ are written with data from $\mathbf{T}$ according to:

$$\mathbf{B}_0[m] = \mathbf{T}[n] \quad \text{for} \quad n = 4m \qquad \mathbf{B}_1[m] = \mathbf{T}[n] \quad \text{for} \quad n = 4m+1$$

$$\mathbf{B}_2[m] = \mathbf{T}[n] \quad \text{for} \quad n = 4m+2 \qquad \mathbf{B}_3[m] = \mathbf{T}[n] \quad \text{for} \quad n = 4m+3$$

$$m \in [0, \frac{L'}{4} - 1], \quad \frac{L}{4} \in Z, \quad L > 0 \tag{6.2.3}$$

and we have four sample-type indices, $p_i \in \{0, 1, 2, 3\}$ with $i \in [0, 3]$.

The phase index, ϕ_I, is transformed into four physical block addresses $\phi_i \in [0, \frac{L}{4} - 1]$ with $i \in [0, 3]$ which respectively address memory blocks $\mathbf{B}_i$ modulo-$\frac{L}{4}$ and generate the sample set $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1], \mathbf{T}[\phi_I + 2], \mathbf{T}[\phi_I + 3]\}$ for $\phi_I \in [0, 2^I - 1]$. It is evident that the block addresses are obtained through modular division by four. ϕ_3 is obtained by taking the integer part of $\frac{\phi_I}{4}$, modulo-$\frac{L}{4}$ (i.e. $\left\langle \left\lfloor \frac{\phi_I}{4} \right\rfloor \right\rangle_{\frac{L}{4}}$), with ϕ_2, ϕ_1 and ϕ_0 obtained by offsetting ϕ_I by integer increments before the modular division operation. Assuming L is exactly divisible by four, the block addresses for the order-4 CAVM are given by:

$$\phi_0 = \left\langle \left\lfloor \frac{\phi_I + 3}{4} \right\rfloor \right\rangle_{\frac{L}{4}} \qquad \phi_1 = \left\langle \left\lfloor \frac{\phi_I + 2}{4} \right\rfloor \right\rangle_{\frac{L}{4}}$$

$$\phi_2 = \left\langle \left\lfloor \frac{\phi_I + 1}{4} \right\rfloor \right\rangle_{\frac{L}{4}} \qquad \phi_3 = \left\langle \left\lfloor \frac{\phi_I}{4} \right\rfloor \right\rangle_{\frac{L}{4}} \tag{6.2.4}$$

Table (6.2.2) illustrates the block address and sample-type index sequences for an order-4 CAVM with $L = 8$.

ϕ_I	ϕ_0	ϕ_1	ϕ_2	ϕ_3	$\mathbf{B}_0[\phi_0]$	p_0	$\mathbf{B}_1[\phi_1]$	p_1	$\mathbf{B}_2[\phi_2]$	p_2	$\mathbf{B}_3[\phi_3]$	p_3
0	0	0	0	0	T[0]	0	T[1]	1	T[2]	2	T[3]	3
1	1	0	0	0	T[4]	3	T[1]	0	T[2]	1	T[3]	2
2	1	1	0	0	T[4]	2	T[5]	3	T[2]	0	T[3]	1
3	1	1	1	0	T[4]	1	T[5]	2	T[6]	3	T[3]	0
4	1	1	1	1	T[4]	0	T[5]	1	T[6]	2	T[7]	3
5	0	1	1	1	T[0]	3	T[5]	0	T[6]	1	T[7]	2
6	0	0	1	1	T[0]	2	T[1]	3	T[6]	0	T[7]	1
7	0	0	0	1	T[0]	1	T[1]	2	T[2]	3	T[7]	0

Table (6.2.2): Order-4 CAVM address sequences, memory block data values and sample-type indices for $L = 8$. *The* $\mathbf{B}_0[\phi_0], \mathbf{B}_1[\phi_1], \mathbf{B}_2[\phi_2]$ *and* $\mathbf{B}_3[\phi_3]$ *columns illustrate sample ordering permutations at the memory block outputs.*

For $\phi_I \in [0, 2^I - 1]$, the order-4 sample-type indices take values that follow a cyclic permutation of the set $\{0, 1, 2, 3\}$. The sample-type indices are obtained by offsetting ϕ_I prior to the modulo-4 operation and then subtracting the result from three. The sample-type indices for the order-4 CAVM are therefore given by:

$$\begin{aligned} p_0 &= 3 - \langle \phi_I + 3 \rangle_4 \qquad p_1 = 3 - \langle \phi_I + 2 \rangle_4 \\ p_2 &= 3 - \langle \phi_I + 1 \rangle_4 \qquad p_3 = 3 - \langle \phi_I \rangle_4 \end{aligned} \tag{6.2.5}$$

Figure (6.2.5) illustrates the arithmetic processing model for an order-4 CAVM and cubic interpolation processor with a fractional phase address $(\phi_I + \alpha)$ and wavetable index ψ.

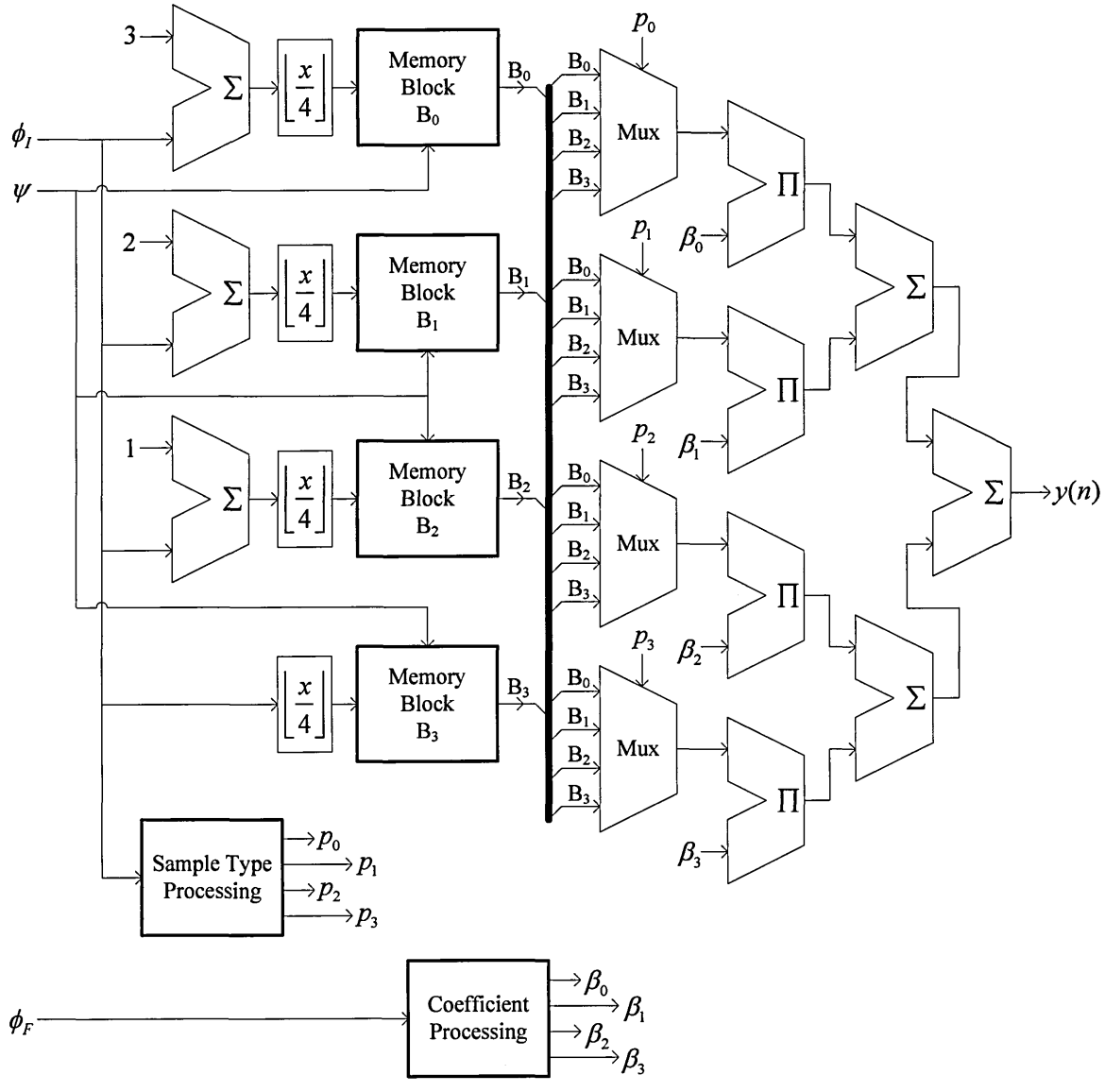


Figure (6.2.5): Order 4 CAVM and cubic interpolation processing model.

The ϕ_i and p_i terms with $i \in [0, k-1]$ are dependent on a modular division by 4 operation which is effected with two consecutive right shift operations and therefore computationally trivial. Furthermore, this holds for all radix 2 values of k requiring $\log_2(k)$ consecutive right shift operations.

6.2.4 The Generalised CAVM and Interpolation Process Model

The preceding discussion indicates a distinct pattern in the attributes of an order-k CAVM which we now formalise. It is evident from section (6.2.3) that the CAVM block addresses and sample-type indices are dependent on a division-by-k operation which is problematic for all *odd*-order CAVM configurations. For all *even* order architectures, this operation is implemented with an appropriate bit-wise right shift. For completeness, we present the order-3 CAVM architecture in Appendix C which includes a method for effecting the required divide-by-3 operation. However, modulo $\frac{L}{3}$ arithmetic is still necessary.

A memory space, $\mathbf{T}$, of length L' samples which contains an integer number of contiguous wavetables each of length L samples contains $\frac{L'}{L} \in Z$ distinct wavetables. An order-k CAVM partitions $\mathbf{T}$ into k distinct memory blocks denoted $\mathbf{B}_i$ with $i \in [0, k-1]$, each of length $\frac{L}{k} \in Z$ samples. We have a fractional addressing representation comprising the phase component $(\phi_I + \alpha)$ and a wavetable index component, ψ, bound according to:

$$\begin{aligned} &\phi_I \in [0, L-1] \\ &\alpha \in [0, 1) \\ &\psi \in [0, \frac{L'}{L} - 1] \\ &\frac{L}{k} \in Z \quad \frac{L'}{L} \in Z \end{aligned} \tag{6.2.6}$$

The contents of the i^{th} CAVM memory block are given by:

$$\mathbf{B}_i[m] = \mathbf{T}[n] \quad \text{for} \quad n = km + i$$

$$m \in [0, \frac{L'}{k} - 1], \quad i \in [0, k-1] \tag{6.2.7}$$

The individual CAVM memory block addresses, ϕ_i, are given by:

$$\phi_i = \left\langle \left\lfloor \frac{\phi_I + (k-i-1)}{k} \right\rfloor \right\rangle_{\frac{L}{k}}$$

$$\phi_I \in [0, L-1], \quad i \in [0, k-1] \tag{6.2.8}$$

The sample-type indices, p_i, are given by:

$$p_i = (k-1) - \langle \phi_I + (k-i-1) \rangle_k$$

$$\phi_I \in [0, L-1], \quad i \in [0, k-1] \tag{6.2.9}$$

An order-k CAVM is associated with an order-$N = (k-1)$ interpolation as described in section (5.2). The interpolation coefficients, $\beta_i(\alpha)$ $i \in [0, N]$, are defined by Eq. (5.2.3) and assume the CAVM is indexed with an offset integer phase component, $\left\langle \phi_I - \left\lfloor \frac{N-1}{2} \right\rfloor \right\rangle_{2^I}$, to provide a minimum interpolation error bound. The fractional phase component, ϕ_F, determines the fractional phase value, $\alpha = \frac{\phi_F}{2^F} \in [0, 1)$, according to Eq. (5.1.5) and ultimately the interpolation coefficients through Eq. (5.2.3). Interpolation coefficients may be computed directly using ϕ_F (and hence α) as an argument, requiring $O(N^2)$ multiplications. However, this incurs a significant computational imposition in real-time applications, particularly when N is large. An alternative technique already mooted in section (5.2.4) replaces direct computation with lookup tables indexed by ϕ_F. We require $k = (N+1)$ β-lookup tables, each tabulating a particular interpolation coefficient according to Eq. (5.2.3) and indexed by

$\phi_F \in [0, 2^F - 1]$ which outputs the β coefficient according to the value of the α argument. Each β-lookup table contains 2^F coefficients and imposes a memory overhead of $(N+1)2^F$ words. The i^{th} coefficient lookup table, $\mathbf{C}_i[a]$ with $i \in [0, k-1]$ and address $a \in [0, 2^F - 1]$, is tabulated according to:

$$\mathbf{C}_i[a] = \prod_{\substack{j=0 \\ j \neq i}}^{N} \left(\frac{\frac{a}{2^F} + \left\lfloor \frac{N-1}{2} \right\rfloor - j}{i - j} \right) \qquad (6.2.10)$$

$$i \in [0, N] \qquad a \in [0, 2^F - 1]$$

where we note that the subscript variable k used in Eq. (5.2.3) has been replaced by l to avoid confusion with the CAVM order denotation we use in this Chapter. Each table lookup operation therefore outputs $\beta_i(\alpha) = \beta_i\left(\frac{\phi_F}{2^F}\right) = \mathbf{C}_i[\phi_F]$.

We extend the utility of the β-coefficient lookup table to effectively integrate the data reordering function and thereby obviate N reordering multiplexers. For radix-2 values of k, the k-sample set of cyclic permutations are uniquely addressed by the $\log_2(k)$ *least significant bits* of the phase index, ϕ_I. We organise each coefficient lookup table into k pages of interpolation coefficient values, with ϕ_F indexing the F least significant bits and $\langle \phi_I \rangle_k$ indexing the $\log_2(k)$ most significant bits to select a particular page. The ordering of the k interpolation coefficients, $\beta_i(\alpha)$ with $i \in [0, k-1]$, within the paginated lookup table reflects the k sample-type permutations from the memory blocks. Table (6.2.3) illustrates $\beta_i(\alpha)$-page allocation across the four coefficient lookup table pages for an order-4 CAVM example case. We denote a paginated coefficient lookup table by the vector $\mathbf{C}_i[\phi_F, g]$, where $g \in [0, k-1]$ denotes the page address. The

total coefficient lookup table memory overhead is now 2^{F+4} words for the order-4 CAVM example and with $F = 12$ the total coefficient lookup table memory overhead is 65536 words.

g	$\mathbf{C}_0[\phi_F, g]$	$\mathbf{C}_1[\phi_F, g]$	$\mathbf{C}_2[\phi_F, g]$	$\mathbf{C}_3[\phi_F, g]$
0	$\beta_0(\alpha)$	$\beta_1(\alpha)$	$\beta_2(\alpha)$	$\beta_3(\alpha)$
1	$\beta_3(\alpha)$	$\beta_0(\alpha)$	$\beta_1(\alpha)$	$\beta_2(\alpha)$
2	$\beta_2(\alpha)$	$\beta_3(\alpha)$	$\beta_0(\alpha)$	$\beta_1(\alpha)$
3	$\beta_1(\alpha)$	$\beta_2(\alpha)$	$\beta_3(\alpha)$	$\beta_0(\alpha)$

Table (6.2.3): Interpolation coefficient lookup table organisation for the order-4 CAVM.

Figure (6.2.6) illustrates an order-4 CAVM process model employing augmented coefficient lookup tables to obviate the reordering multiplexers. In general, for an order-k CAVM employing this technique, the coefficient lookup table address is $\left(\phi_F + 2^F \langle \phi_I \rangle_k\right) \in \left[0, \left(2^F k\right) - 1\right]$, imposing a total memory overhead of $2^F k^2$ words and requiring $\lceil \log_2(k) \rceil$ extra address bits (e.g. 2 additional bits for $k = 3$ and $k = 4$). In a physical implementation, the k-fold coefficient memory increase must be assessed relative to the cost of k k-to-1 multiplexers and the associated bus connectivity as shown in Figure (6.2.5) for the $k = 4$ case.

Truncating the phase fraction field, ϕ_F, by R bits as discussed in section (5.1.2), allows SNR performance to be exchanged for coefficient lookup table length. The total

memory overhead is now $2^{F-R}k^2$ words and decreases exponentially with R. Figure (6.2.7) illustrates the variation in SNR with phase fraction truncation using a Mathcad model given in Appendix B for a 4096 location wavetable tabulating a signal composed of 100 harmonics with -3 dB/octave roll-off slope as depicted in Figure (5.4.4), with $M = 24$, $I = 12$ and $\varphi = 5715$.

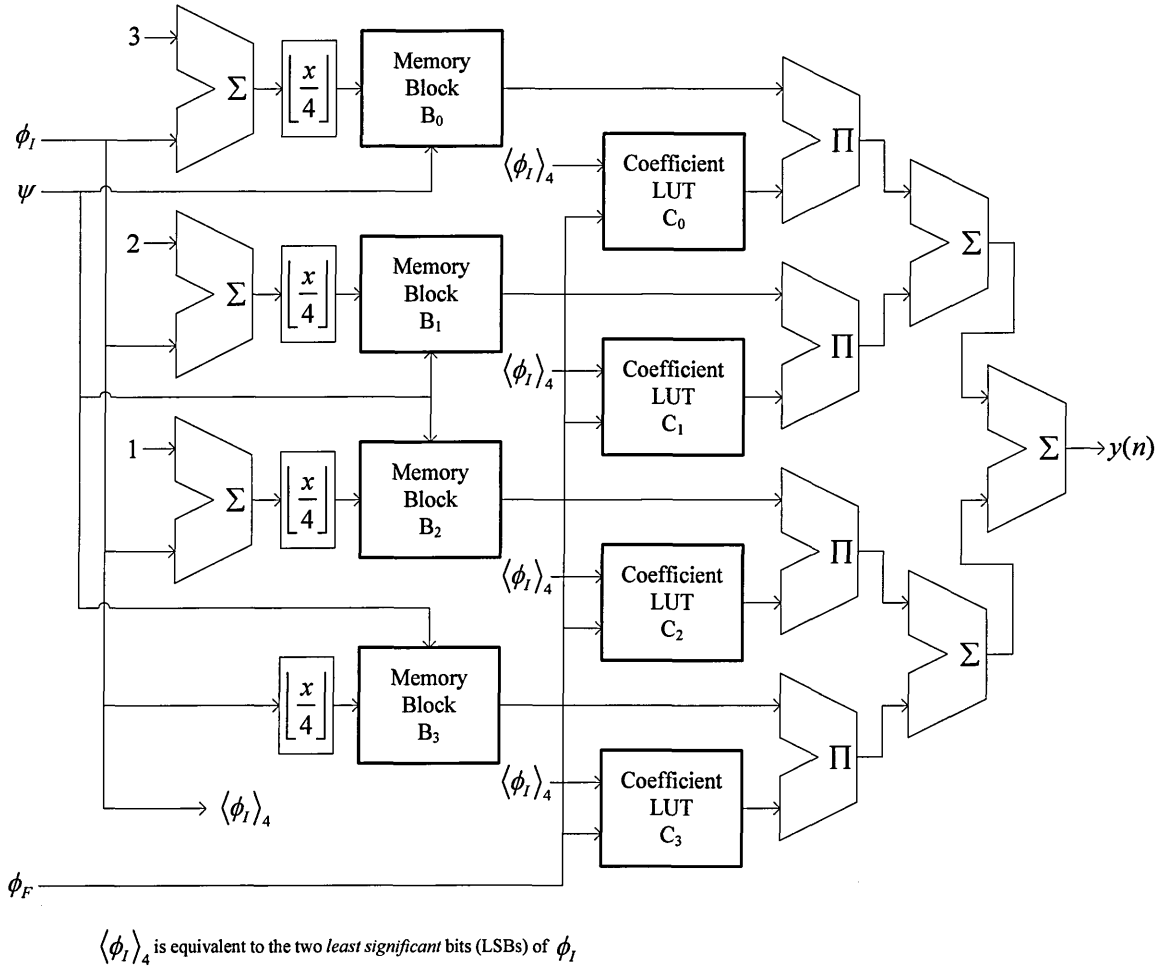


Figure (6.2.6): An order-4 CAVM process model using augmented coefficient lookup tables to eliminate reordering multiplexers.

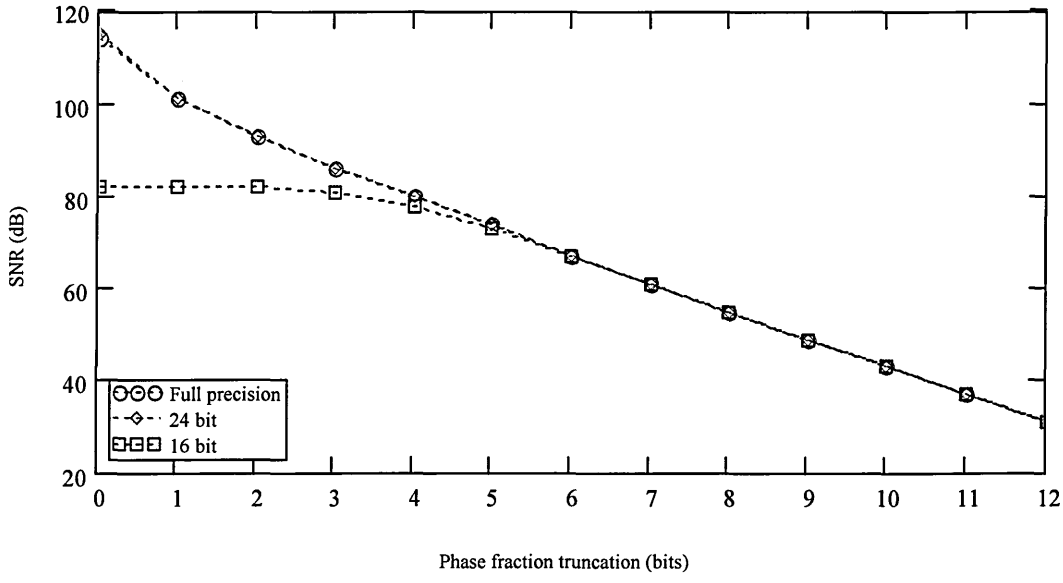


Figure (6.2.7): Variation of SNR with phase fraction truncation and three levels of arithmetic precision.

The simulation indicates that with 16-bit fixed-point arithmetic and $M = 24$, approximately eight of the most significant fraction bits are required to maintain SNR for this spectrum. Hence, the coefficient lookup table memory is reduced sixteen-fold in this case compared to the brute force case when $F = 12$.

6.2.5 Linear Wavetable Combination

Hitherto we have considered fractional *phase* interpolation associated with phase accumulating frequency synthesis of a tabulated signal. We can extend the fractional addressing model to include linear combinations of multiple *wavetables* in line with the SIS and MWS processing models presented in Chapter 2. If we consider the SIS model, our wavetable indexing sequence denoted by ψ (see Figure (6.2.1) and once again dropping the time-index for brevity) becomes a *fractional* quantity with integer and fraction components denoted by $\psi_{I'}$ and $\psi_{F'}$, respectively where I' and F' denote the

respective integer and fraction field widths in bits. Hence, we have a wavetable fractional index, λ, given by:

$$\lambda = \frac{\psi_{F'}}{2^{F'}} \in [0,1) \tag{6.2.11}$$

The SIS model amounts to fractional addressing at the *wavetable* level (i.e. linear interpolation between wavetables) as distinct from the *phase* level previously considered. Assuming multiple consecutive wavetables stored in the vector $\mathbf{T}$, we denote an order-N interpolated output sample by $\mathbf{T}[\phi, \psi_{I'}]_N$, where ϕ denotes the sample *phase* index and $\psi_{I'}$ denotes the wavetable index. Linear interpolation between *consecutive* wavetables executing the SIS processing model is then given by:

$$y(n) = \mathbf{T}[\phi, \psi_{I'}]_N + \lambda(\mathbf{T}[\phi, (\psi_{I'}+1)]_N - \mathbf{T}[\phi, \psi_{I'}]_N)$$
$$\lambda \in [0,1) \tag{6.2.12}$$

where $y(n)$ denotes the interpolated output sample sequence. Eq. (6.2.12) interpolates between two consecutive wavetables from the set of $2^{I'}$ possible wavetables contained in $\mathbf{T}$ and indexed by $\psi_{I'}$ and $(\psi_{I'}+1)$.

The order-N interpolated output sample $\mathbf{T}[\phi, \psi_{I'}]_N$ is implemented with an order-$(N+1)$ CAVM and interpolation processing as presented in section (6.2). We extend the CAVM paradigm to the wavetable level as illustrated in Figure (6.2.8) which linearly interpolates between two consecutive wavetables stored within two CAVM blocks. Even numbered wavetables are stored in the first CAVM block and odd numbered wavetables in the other. A single data-parallel CAVM read operation produces the $\mathbf{T}[\phi, \psi_{I'}]_N$ and $\mathbf{T}[\phi, (\psi_{I'}+1)]_N$ data set which is fed to a linear interpolation block in a similar manner to that of the order-2 CAVM interpolator discussed in section (6.2.2). The sample reordering multiplexers are controlled from the

least significant bit of the integer wavetable index and its Boolean complement which we denote by $\psi_{I'(\mathrm{LSB})}$ and $\overline{\psi_{I'(\mathrm{LSB})}}$.

We generalise this concept by considering the linear combination of multiple wavetables in line with the MWS model. However, this generalisation only has utility if the k wavetables in any given "MWS set" comprise k *consecutive* wavetables held in memory. Additionally, the MWS model requires k distinct weighting coefficients to effect a linear combination. Polynomial interpolation of k consecutive wavetables from a *single* fractional address is theoretically possible, but of less utility.

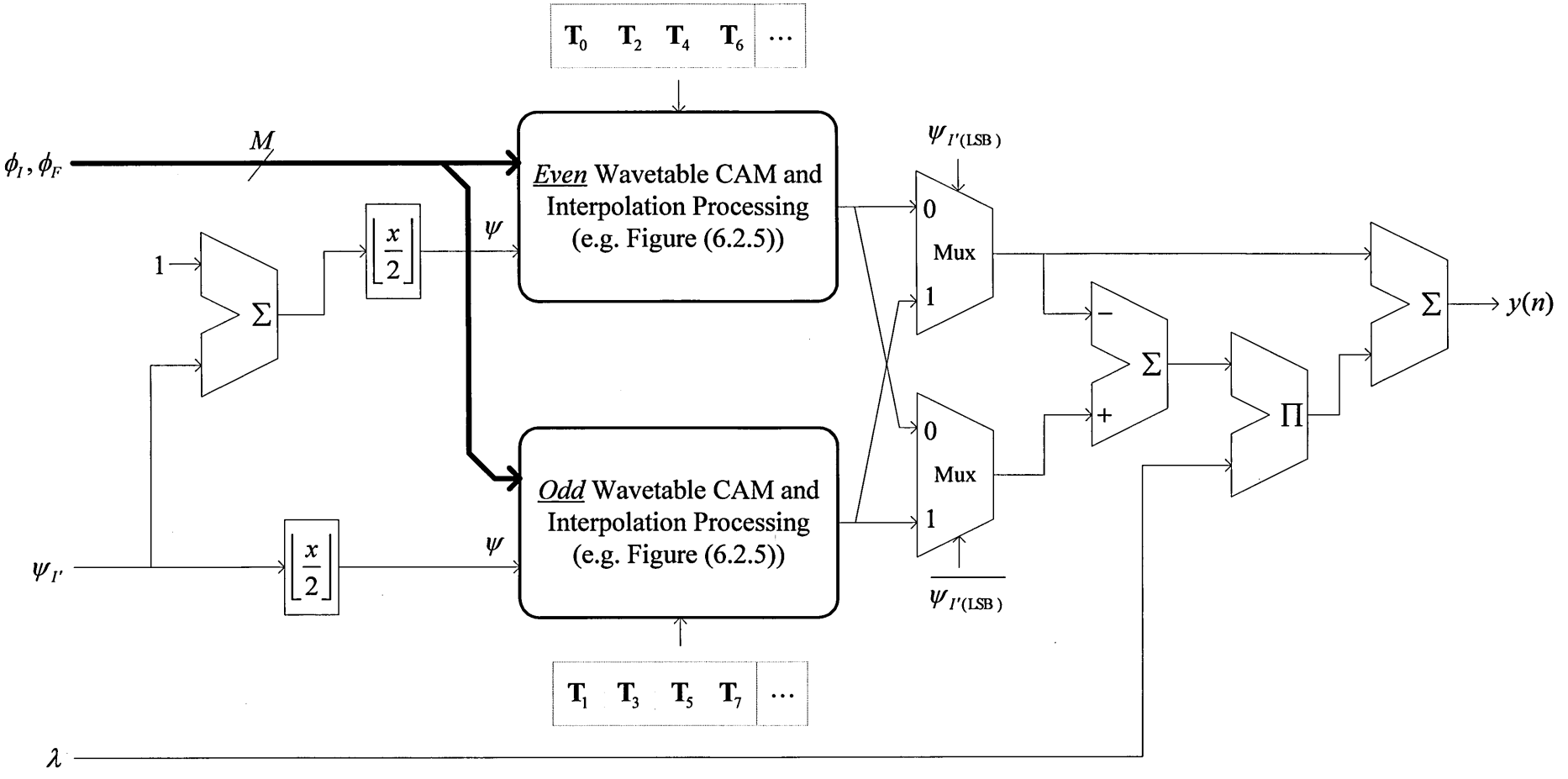


Figure (6.2.8): Linear interpolation between two consecutive wavetables executing the SIS processing model. Odd and even wavetables are stored in respective CAM blocks providing efficient execution of the linear wavetable interpolation according to the fractional address, λ.

6.3 Phase Domain Processing

6.3.1 Introduction

In this section we develop the concept of phase domain processing to execute the HAS and PAS processing models. This material builds on section (4.2.5) where we reviewed phase control of the phase accumulating sinusoidal oscillator. We begin by reviewing the concept of *block pipelining* to enhance throughput in algorithms with highly sequential arithmetic processes such as the HAS and PAS models. Block pipelining is readily applicable to the processing models presented in section (6.2) to compute multiple voice WLS realisations. The block pipelined phase accumulator which we use as an example, underpins a multi-voice PAS processing model presented in section (6.3.7).

6.3.2 Block Pipelining and the Phase Accumulating Oscillator

The *block pipelining* technique is an extension of the classical single sample pipelining model used to enhance computational performance of single sample processing systems [Pirsch, 1996]. Single sample pipelines partition the processing chain with registers to enable the processing function of each stage to execute in a single clock interval. Block pipelines adopt a similar architecture, but now the registers are replaced by dual-port memory (DPM) elements. A DPM is characterised by a single memory space accessible via two distinct access ports, each comprising their own data and address busses. In a block pipeline, *complete blocks* of data are processed *en bloc* as they "propagate" down the pipeline in a systolic fashion. Memory contention[4] is prevented by arranging the MSBs of the two address ports to be mutually complementary and toggling at the block processing rate (i.e. typically the system sample rate). This technique is known as "ping-

pong" or double buffering in the literature [Symons, 1995; Ackenhusen, 1999]. When one half of the DPM space is being written with data, the other half is simultaneously read and vice versa, thereby avoiding memory access contention at the expense of introducing a latency of one block processing cycle. As with single sample pipelines, block pipelines can have I/O ports introduced at any point along the pipe, observing a reduced latency for these ports. Figure (6.3.1) shows a simple block pipeline process model which partitions two hypothetical processes denoted by the functions F and G.

[4] A memory contention occurs when one memory location is being written by one port simultaneous with being read or written by the *other* port leading to the possibility of erroneous data transfer.

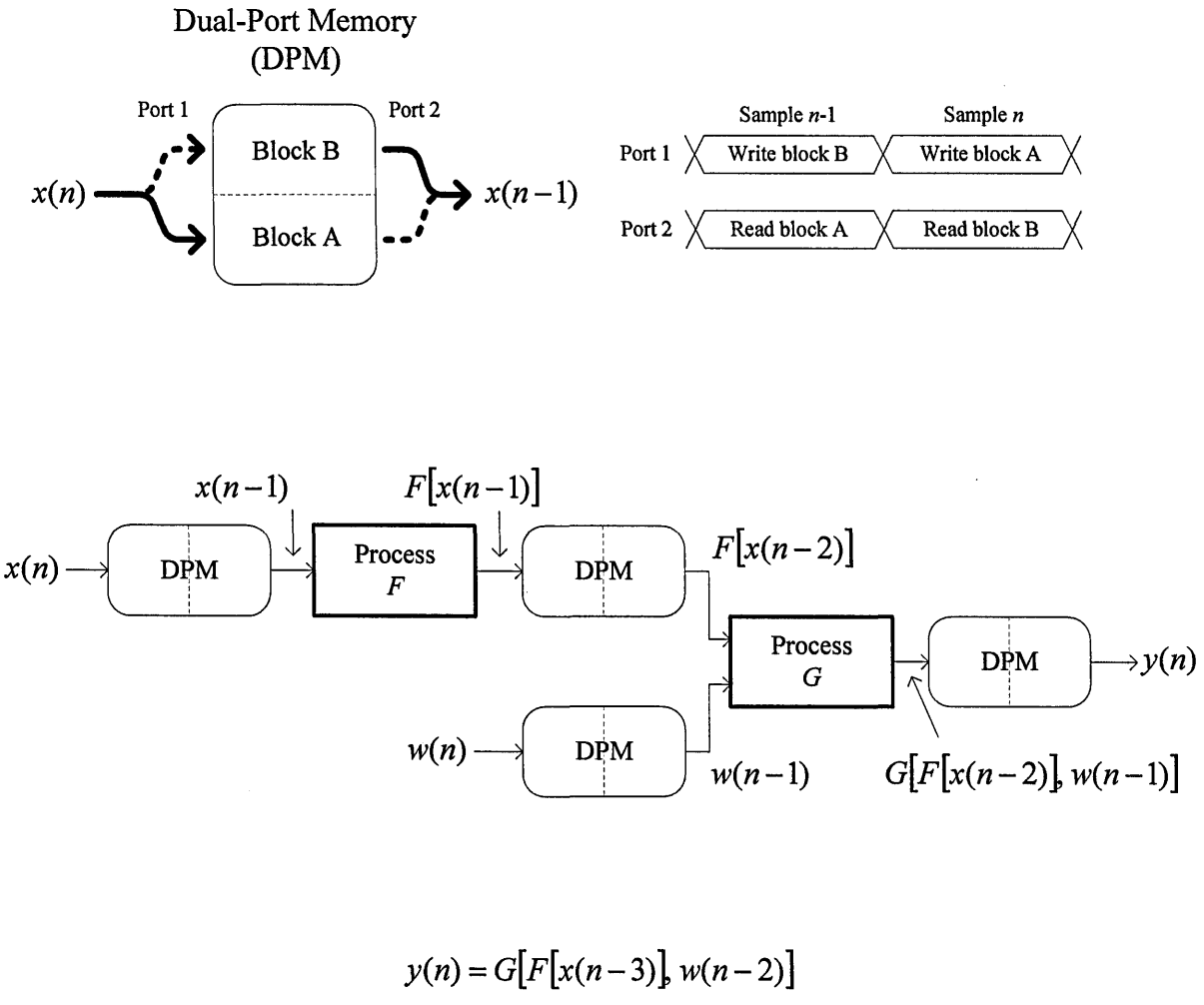


*Figure (6.3.1): Rudimentary **block** pipelining process model using "ping-pong" dual-port memory (DPM) to partition processing elements.*

In addition to partitioning the pipelined execution of a particular process model, ping pong DPM allows updating of control parameter data to be completely decoupled from the computational process albeit at the cost of one sample latency. While the pipelined processor is consuming parametric data, the next parameter blocks are loaded without interrupting computation flow. This architecture therefore readily lends itself to a memory-mapped coprocessor model within a host computer system.

We illustrate the utility of a block pipeline processing model by considering a time-division multiplexed phase accumulator synthesising multiple sinusoidal partial oscillators as illustrated in Figure (6.3.2). Three *control parameter* DPM blocks hold partial phase increment, start phase and amplitude parameters and may be updated from a host control computer through appropriate memory mapping of the DPM input ports into the host memory space. Effecting DPM ping-pong switching every sample period enables control parameter updates at the system sample rate. Three *state-variable* DPM blocks (shown shaded) partition time-critical computations within the processing pipeline and provide state-variable storage between processing elements. The first state-variable DPM holds accumulated phase values and pipelines the phase accumulation operation. A second DPM stores phase-mapped sine samples and pipelines the phase offset addition and sinusoidal phase mapping operations. Finally, a third DPM stores amplitude-weighted sine samples ahead of further processing and so pipelines the amplitude multiply operation.

Block pipelining requires that within a particular pipeline stage, we compute N_p elemental partial operations (e.g. phase accumulation) every sample period. Accordingly, we sub-divide each sample period into N_p time-slots of equal duration $\frac{1}{N_p f_s}$ seconds, in which elemental operations for *each* of N_p oscillators are executed.

We therefore envisage a *time-slot address* (TSA) which partitions the sample period into N_p distinct, equal duration time-slots with TSA $\in [0, N_p - 1]$ and addresses all but the most significant bit (MSB) of both DPM address ports. The TSA is typically generated by a counter clocked at $N_p f_s$ whose range must span half the DPM address space. We denote the time-multiplexed parameters and state variables with the subscript j corresponding to a particular TSA value that uniquely identifies each partial and so $j \in [0, N_p - 1]$.

It is evident that block pipelining causes increased global latency within the arithmetic process flow and a differential latency between the three control parameters and the oscillator output samples, $y_j(n)$, as illustrated in Figure (6.3.3). We have:

$$y_j(n) = A_j(n-2)\mathrm{S}\left[\phi_j(n-3) + \Phi_j(n-3)\right] \qquad (6.3.1)$$

where $\phi_j(n) = \left\langle n\varphi_j(n-1)\right\rangle_{2^M}$. Eq. (6.3.1) shows that there is a three sample latency for the start phase parameter, a two sample latency for the amplitude parameter and a four sample latency for the phase increment parameter. This latency skewing can be corrected if necessary by inserting appropriate delays in the computation of control parameters with least latency (i.e. by delaying the amplitude and start phase parameters in this example).

We now proceed to investigate arithmetic processing architectures which execute the HAS and PAS processing models in real-time using block and single-sample pipelining to enhance throughput. To improve clarity in subsequent process model development we do not show DPM blocks within the signal processing path. However, we show control parameter memories as DPM blocks with implicit ping-pong functionality. DPM blocks may be inserted at any point in the processing path to decouple process elements and thereby block pipeline the arithmetic computations.

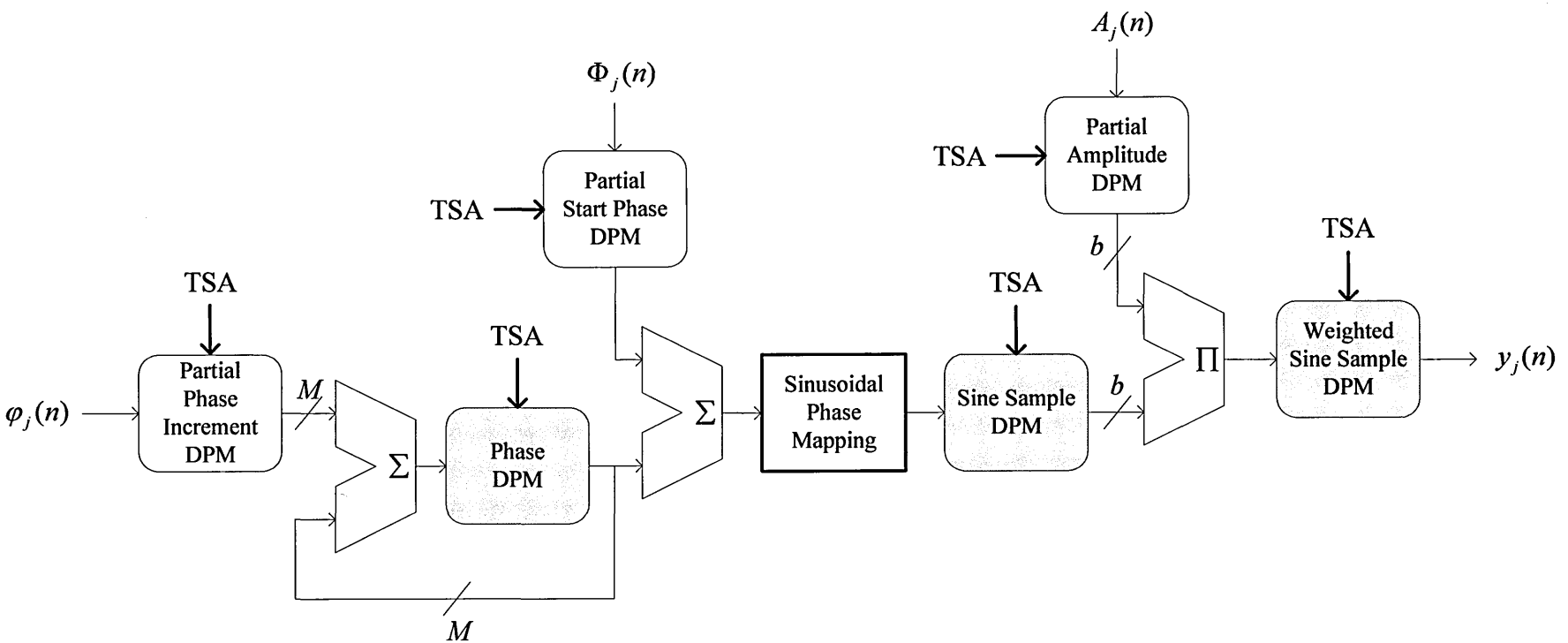
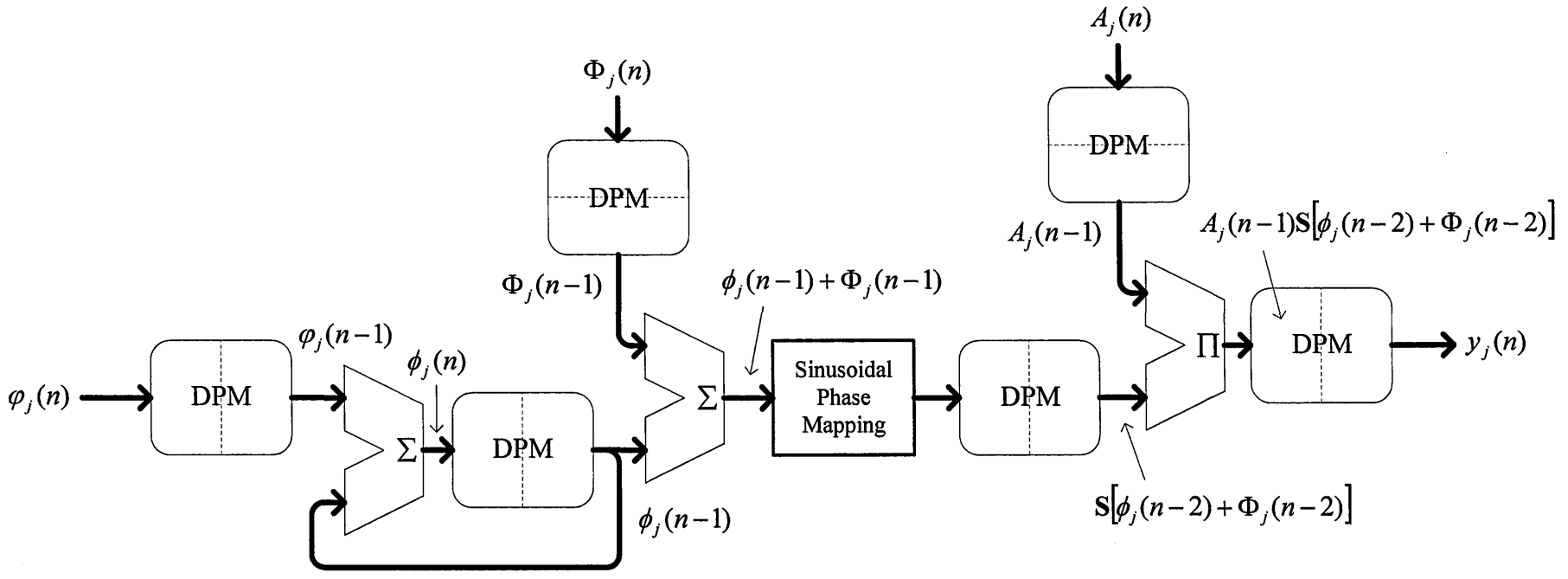


Figure (6.3.2): Multiplexed phase-accumulator process model synthesising multiple, independently controlled partials. (Shaded DPM blocks denote state-variable storage elements.)



$$\phi_j(n) = \langle n\varphi_j(n-1) \rangle_{2^M}$$

$$y_j(n) = A_j(n-2)\mathbf{S}[\phi_j(n-3)+\Phi_j(n-3)] = A_j(n-2)\mathbf{S}[\langle n\varphi_j(n-4) \rangle_{2^M} + \Phi_j(n-3)]$$

Figure (6.3.3): Block-pipelined signal flow for the synthesis of multiple partials illustrating parameter latency.

6.3.3 Pitch Control in the Phase Accumulating Oscillator

The phase accumulating digital oscillator executes a discrete-time integration of phase increment, φ, using an M-bit accumulator and generates a phase sequence, $\phi(n)$, which is phase-mapped to synthesise a corresponding amplitude sequence. For a sample rate denoted by f_s, the oscillation frequency is given by $\frac{\varphi f_s}{2^M}$ and is *linearly proportional* to the phase increment parameter φ. In section (4.2.2) we reviewed the frequency control resolution required for musical synthesis and the nature of pitch and equally-tempered tuning. Building on this material, we use a lookup table as illustrated in Figure (6.3.4) to translate between a *pitch* control parameter, ρ, which is characterised by the number of semitones per bit change and *phase increment* (i.e. a frequency control parameter) which is characterised by the number of Hertz per bit change.

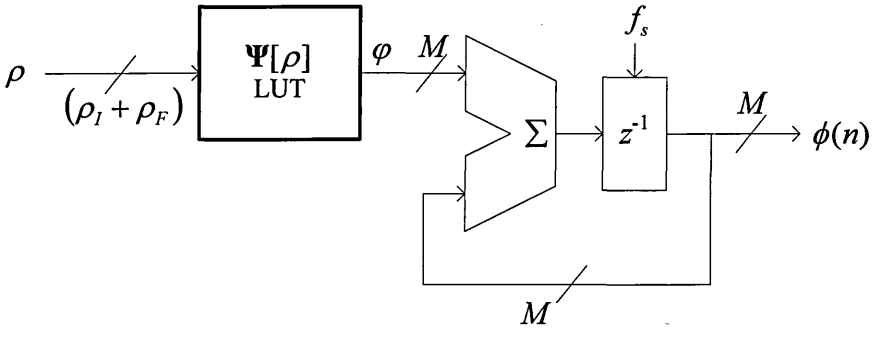


Figure (6.3.4): Phase accumulator incorporating lookup table to effect pitch control.

ρ is a fixed-point fractional quantity whose fraction field, F_ρ, determines the pitch tuning *resolution* in fractions of a semitone and whose integer field, I_ρ, determines the tuning *range* in semitones. A ρ value represented by 14 bits and partitioned equally into 7 integer and fraction bits, provides a tuning *range* of 127 semitones (or just over 10 octaves) and a tuning *resolution* of $\frac{1}{128}$ semitone (i.e. slightly better than 1 cent).

In general, for a tuning range and fraction resolution denoted by R and r *semitones*, respectively, we have $I_\rho = \lceil \log_2(R) \rceil$ and $F_\rho = \left\lceil -\log_2\left(\frac{1}{r}\right) \right\rceil$ bits.

The pitch-to-phase increment lookup table which we denote by the vector $\boldsymbol{\Psi}$, contains $2^{(I_\rho + F_\rho)}$ M-bit *integer* values and is tabulated over a range of address values, a, according to:

$$\boldsymbol{\Psi}[a] = \left\lfloor \frac{B\gamma^a 2^M}{f_s} + 0.5 \right\rfloor$$
$$a \in [0, 2^{(I_\rho + F_\rho)} - 1] \tag{6.3.2}$$

where B represents the baseline frequency[5] when $a = 0$ (i.e. $\rho = 0$) and γ represents the *minimum* equally tempered tuning frequency ratio (i.e. pitch resolution) corresponding to a *single* least significant bit change in the table address and hence ρ. For a pitch control resolution of $\frac{1}{128}$ semitone within the equally tempered tuning system, we have $\gamma = 2^{\frac{1}{12(128)}}$ (i.e. $\gamma^{128} = \sqrt[12]{2}$ – the *semitone* frequency ratio within the equally tempered scale). A *constant pitch* offset can be imposed on the oscillation frequency by adding a transposition parameter, β [6], to the lookup table input argument ρ. Obtaining the same pitch transposition in the "phase increment" (i.e. frequency) domain requires *multiplicative* scaling of φ by β to M-bit precision, with a significant hardware cost associated with an M-bit multiplier, particularly when M is large. We exploit this property in section (6.3.6) where we introduce a technique for synthesising partial phase sequences with arbitrary frequency distribution. Finally, we observe that

[5] The value of B is determined by the *lowest* synthesised pitch required (e.g. 16.35 Hz equivalent to C0).
[6] Not to be confused with the interpolation coefficient denotation used in Chapter 5.

Eq. (6.3.2) *rounds* the tabulated phase increment to the nearest integer value causing a maximum frequency error of $\frac{f_s}{2^{M+1}}$ Hz.

6.3.4 Synthesising Consecutive Harmonic Phase Sequences

We now consider arithmetic processing of the phase accumulator output sequence to generate new phase sequences with harmonic frequency distributions. We generalise this as *phase domain processing*, building on material presented in section (4.2.5) and computational structures proposed by Chamberlin [1976, 1985]. Recalling Eq. (4.2.33), we see that a phase sequence $\phi(n)$ multiplied modulo-2^M by an *integer* k produces a new sequence $\phi'(n)$ whose frequency is exactly the k^{th} harmonic of the $\phi(n)$ frequency, with $\phi'(n) = \langle k\phi(n) \rangle_{2^M}$. Denoting the number of harmonics to be synthesised by N_h, it is evident that there are essentially two approaches to synthesising $\phi'(n)$. One requires multiplicative scaling of φ by k prior to phase accumulation and is undesirable for M values consistent with the requirements of music synthesis (e.g. $M = 24$) since this multiplication must be executed N_h times. An alternative technique illustrated in Figure (6.3.5), utilises a second "phase multiplying" accumulator to provide a *time-multiplexed* sequence of contiguous integer (harmonic) multiples of an input phase sequence, $\phi(n)$, modulo-2^M [Chamberlin, 1976]. This technique exploits the property of a digital accumulator initialised with x to produce the sequence $x, 2x, 3x, \ldots kx$ as the accumulation proceeds to k iterations. The harmonic phase multiplying accumulator is clocked at $N_h f_s$ (where f_s represents the $\phi(n)$ sample rate) and produces the time-multiplexed phase sequence:

$$\phi'(m) = \phi(n-1), 2\phi(n-1), 3\phi(n-1), \cdots, N_h\phi(n-1) \qquad (6.3.3)$$

where we use the time index m to reflect the higher sample rate and observe a one sample pipeline delay. The harmonic multiplying accumulator is loaded with the $\phi(n-1)$ value at the beginning of a sample cycle as depicted in the timing diagram of Figure (6.3.6).

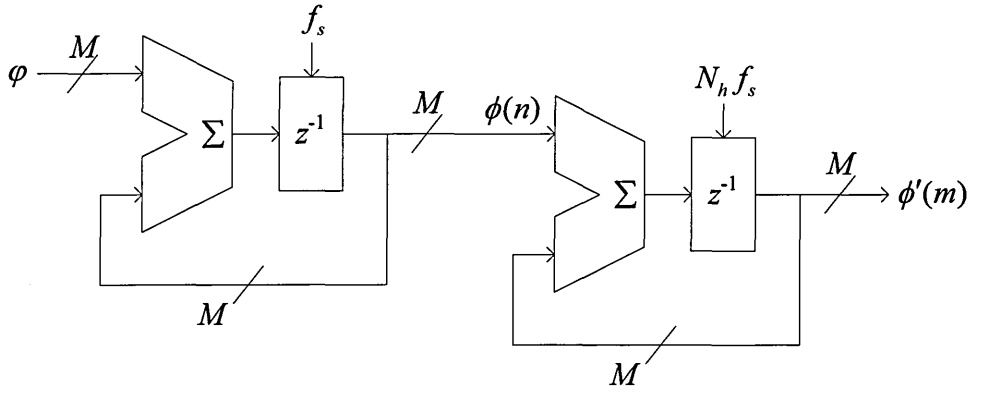


Figure (6.3.5): Generating a time-multiplexed phase sequence, $\phi'(m)$, having a contiguous harmonic frequency distribution.

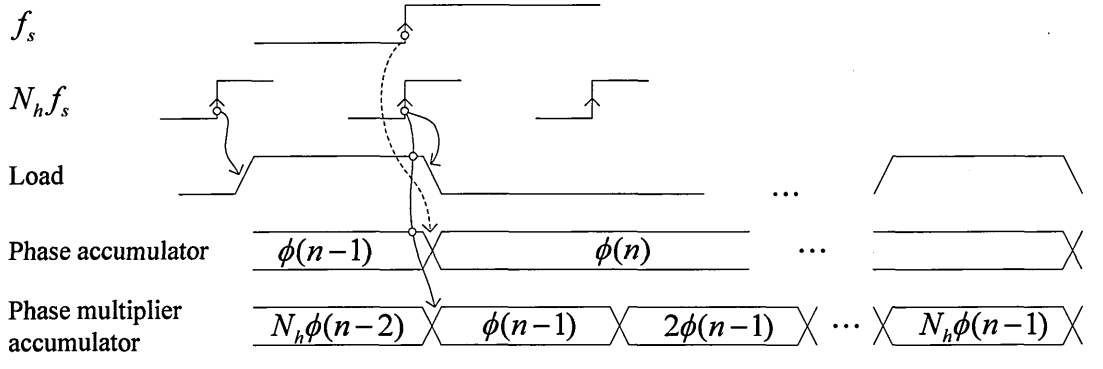


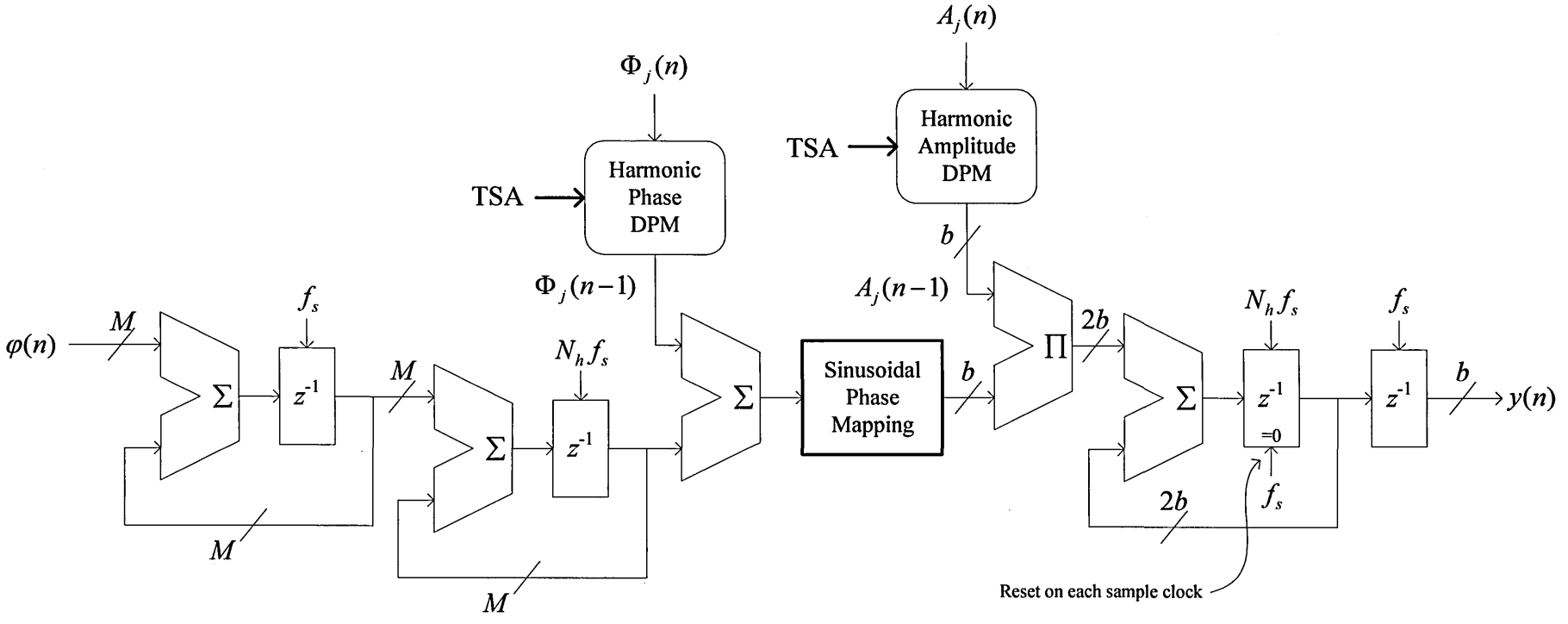
Figure (6.3.6): Timing diagram illustrating initialisation of the harmonic phase multiplier accumulator at the beginning of a sample cycle.

The process model of Figure (6.3.7) sub-divides the sample period into N_h time-slots (similar to the TSA variable in the block pipelining discussion of section (6.3.2)). Each time-slot computes a particular $k\phi(n-1)$ harmonic value which is phase mapped and multiplied by the corresponding amplitude value, $A_j(n)$, according to the HAS

processing model of Eq. (2.3.6)[7]. The N_h weighted harmonic samples are accumulated in a second amplitude accumulator as illustrated in Figure (6.3.7), where DPM blocks supply harmonic amplitude and phase parameter values, $A_j(n)$ and $\Phi_j(n)$, respectively. Aliasing of higher frequency harmonics is prevented by ensuring that N_h is bound so that $\frac{N_h\varphi}{2^M} \le \frac{1}{2}$ and hence $N_h \le \frac{2^{M-1}}{\varphi}$. This may be implemented by dynamically limiting the number of clock cycles applied to the harmonic phase multiplying accumulator as a function of φ.

The phase multiplying accumulator (denoted by "harmonic phase multiplier and offset" in Figure (6.3.7)) efficiently computes *all* harmonics in a consecutive sequence with harmonic multiplier ranging over the interval $[1, N_h]$, where N_h represents the highest harmonic multiplier. This sequence must be computed to the highest required harmonic, irrespective of some intermediate harmonics being superfluous in a typical synthesis application. An unwanted j^{th} harmonic must be excluded by setting the corresponding amplitude parameter to zero effectively wasting computation cycles for each harmonic with $A_j(n) = 0$. In section (6.3.5) we present a modified architecture which allows arbitrary harmonic groupings to be computed.

[7] Here we use the subscript denotation j to avoid confusion with the harmonic multiplying factor k.



$$y(n) = \sum_{j=1}^{N_h} A_j(n-1)\cos\left[2\pi j \frac{\varphi(n-2)}{2^M} n + \Phi_j(n-1)\right]$$

Figure (6.3.7): Implementation of the HAS arithmetic process model using phase domain processing. This architecture depicts a ***single voice*** *synthesiser linearly combining N_h* ***consecutive*** *harmonics into a sample stream, $y(n)$.*

In line with the conclusions from Chapter 5 for sinusoidal WLS, the sinusoidal phase-amplitude mapping block (which is common to all the processing models presented in section (6.3)) executes a linearly interpolated table lookup model. The phase-mapping block uses a first-order difference table to provide data parallelism and ensure the interpolated sample is computed with a *single* parallel table lookup operation as illustrated in Figure (6.3.8).

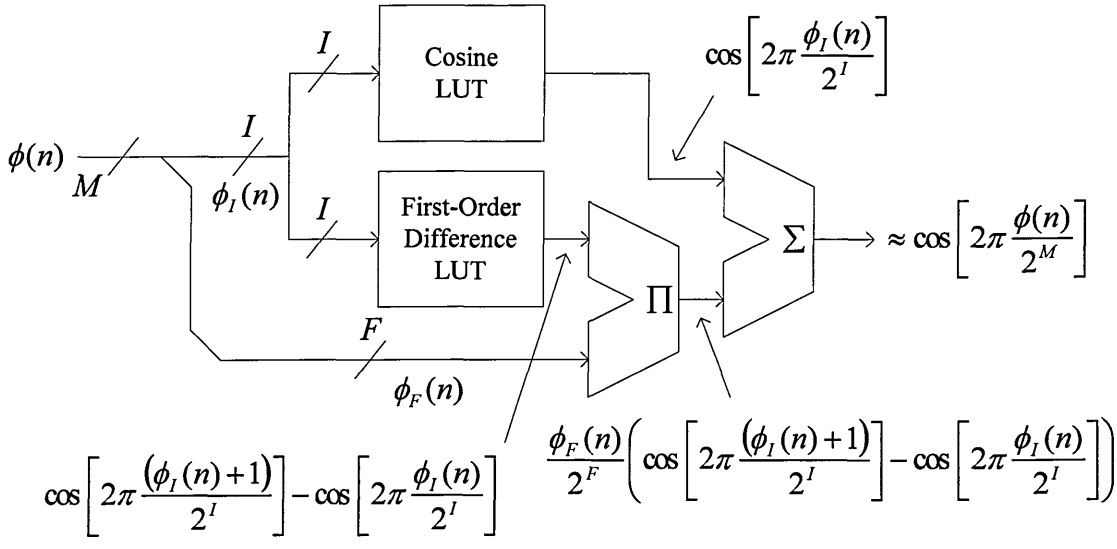


Figure (6.3.8): The linearly interpolated phase-mapping process model using a first-order difference table to eliminate consecutive table lookup operations.

The first-order difference is given by $f(n)=\cos\left[2\pi\frac{(\phi_I(n)+1)}{2^I}\right]-\cos\left[2\pi\frac{\phi_I(n)}{2^I}\right]$ for $\phi_I(n)\in[0,2^I-1]$ and hence the lookup table which we denote by the vector $\mathbf{F}$ is tabulated according to:

$$\mathbf{F}[a]=\cos\left[2\pi\frac{(a+1)}{2^I}\right]-\cos\left[2\pi\frac{a}{2^I}\right] \qquad (6.3.4)$$

$$a\in[0,2^I-1]$$

It is evident that the interpolation multiplier requires asymmetrical operand word sizes which may realise cost savings in VLSI implementations. The phase fraction operand,

$\phi_F(n)$, requires F bits, yet the first-order difference operand will require *less than* the b bits which represent the individual sinusoid samples. We assess the word size required to represent the first-order difference samples by observing that $\max(|f(n)|) \approx \sin\left(\frac{2\pi}{2^I}\right) \approx \frac{2\pi}{2^I}$ for $2^I >> 1$. Since $b-1$ bits represent the sample magnitude in a fixed-point 2's complement representation, the required word size for the first-order difference samples is therefore $\left\lceil \log_2\left(2^{b-1}\left(\frac{2\pi}{2^I}\right)\right)\right\rceil + 1$ bits for $2^I >> 1$. For example, with $I = 10$ and $b = 16$ (i.e. 16-bit fixed point number representation) we require only 9 bits to represent the first-order difference samples.

6.3.5 Synthesising Non-Consecutive Harmonic Phase Sequences

In harmonic (or indeed partial) additive synthesis it is rarely necessary to synthesise a large *consecutive* harmonic set. Typically, groups of harmonics are required with arbitrary (i.e. non-consecutive) frequency distributions [Sandell, 1994]. Extending the harmonic phase multiplying accumulator concept, we introduce an *integer* multiplier block which computes the integer "*k*-tuple", $\phi'_j(n)$, of a phase sequence, $\phi_j(n)$, according to an *arbitrary* harmonic multiplier variable, k_j, thus:

$$\phi'_j(n) = \langle k_j \phi(n) \rangle_{2^M} = \langle n k_j \varphi(n) \rangle_{2^M}$$

$$k_j \in [1, N_h] \quad \phi(n) \in [0, 2^M - 1] \qquad (6.3.5)$$

$$j \in [0, N_c - 1]$$

This approach requires us to discriminate between the maximum harmonic multiplier, $\max(k_j) = N_h$, and the number of harmonics computed per sample period, N_c, with $N_c < N_h$. For example, with $N_c = 4$ we can compute the k sequence $\{1, 3, 7, 11\}$,

whereas with the contiguous processing model of section (6.3.4) we are constrained to the k sequence $\{1, 2, 3, 4\}$.

Figure (6.3.9) illustrates a modified HAS processing architecture where the phase multiplying accumulator is replaced by an *integer multiplier* block which effects multiplication of the phase sequence and an additional DPM block which supplies the k_j integer multiplier operand. The sample period is now partitioned into N_c contiguous computation time-slots by the TSA variable (as outlined in section (6.3.2)) wherein each slot has an arbitrary harmonic multiplier k_j with $j = \text{TSA} \in [0, N_c - 1]$ denoting the particular time-slot. There is no longer an explicit association between TSA value and harmonic multiplier, instead the TSA *indirectly* specifies the arbitrary multiplier (k_j) by addressing the harmonic multiplier DPM. The integer multiplier clearly requires asymmetrical operand word sizes as evident from Eq. (6.3.5) wherein we typically observe $N_h < 256$ and $M = 24$. Hence, an 8 by 24-bit integer multiplier is a reasonable expectation for this arithmetic function. Since we are concerned with integer operands and a product which is always constrained to modulo-2^M, it is evident that a simple multiplier architecture is feasible motivated by modular arithmetic rules from number theory [Weisstein, 1999a]. Specifically, we consider the modular arithmetic *reducibility* and *distributivity* rules, defined thus:

$$\begin{aligned} \langle a \pm b \rangle_n &= \langle \langle a \rangle_n \pm \langle b \rangle_n \rangle \\ \langle ab \rangle_n &= \langle \langle a \rangle_n \langle b \rangle_n \rangle \end{aligned} \tag{6.3.6}$$

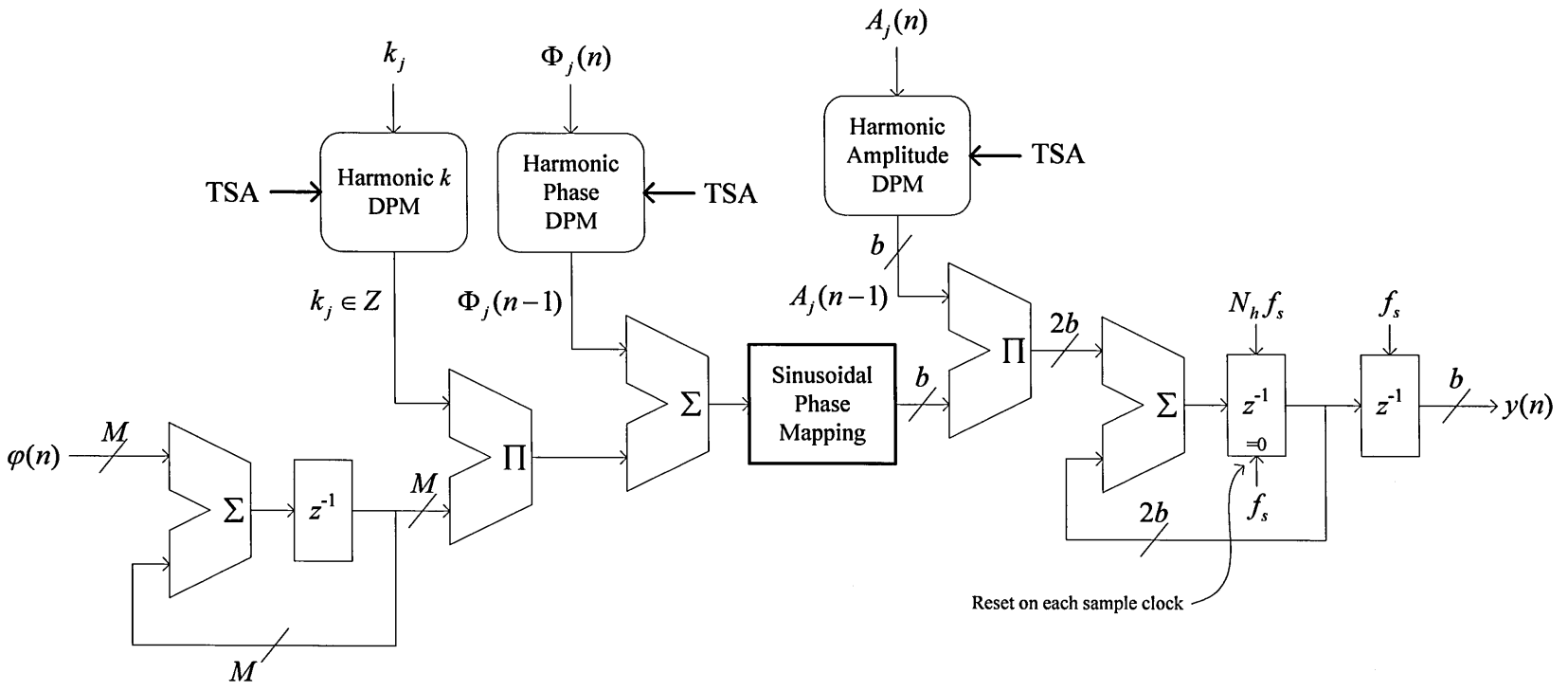
Eqs. (6.3.6) allow us to express Eq. (6.3.5) in a form which may be implemented with a series of shift and add operations which are simple to realise in hardware, thus:

$$\left\langle k_j \phi(n) \right\rangle_{2^M} = \left\langle k_{(0)} \left\langle \phi(n) \right\rangle_{2^M} + k_{(1)} \left\langle 2\phi(n) \right\rangle_{2^M} + k_{(2)} \left\langle 4\phi(n) \right\rangle_{2^M} \ldots k_{(w)} \left\langle 2^w \phi(n) \right\rangle_{2^M} \right\rangle_{2^M}$$

$$= \left\langle \sum_{i=0}^{w} k_{(i)} \left\langle 2^i \phi(n) \right\rangle_{2^M} \right\rangle_{2^M} \tag{6.3.7}$$

$k_{(i)} \in \{0, 1\}$

where $k_{(i)}$ denotes the i^{th} bit of the $w = \lceil \log_2(N_h) \rceil$ bit harmonic multiplier word k and we observe that $k_j \in [0, 2^w - 1]$ using this expression. Eq. (6.3.7) may be readily implemented in hardware using only left-shift, multiplexer and addition operations as illustrated in Figure (6.3.10) for the specific case when $w = 8$. We see that this processing model may be readily pipelined with registers partitioning the time consuming addition stages. The modulo-2^M left-shift operations may be hardwired and incur no hardware overhead as such. In a pipelined implementation, the computation time is limited only by the speed of the individual adder elements with a latency of two clock cycles.



$$y(n) = \sum_{j=1}^{N_h} A_j(n-1)\cos\left[2\pi k_j \frac{\varphi(n-1)}{2^M} n + \Phi_j(n-1)\right] \qquad k_j \in Z, \quad k_j \geq 0$$

Figure (6.3.9): Implementation of the HAS arithmetic process model using phase domain processing. This architecture depicts a ***single voice*** *synthesiser linearly combining N_h harmonics, with* ***arbitrary*** *harmonic multiple according to k_j.*

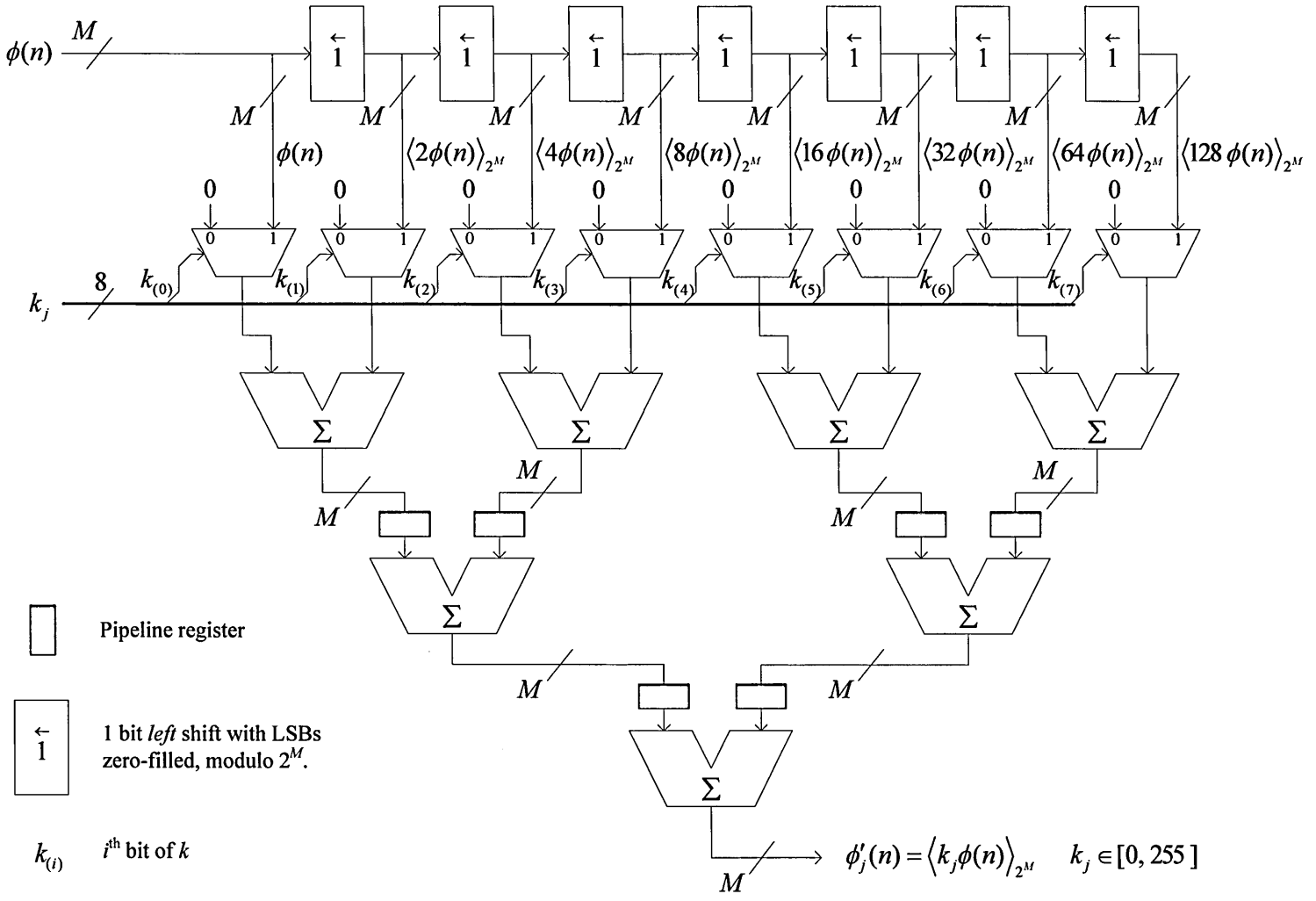


Figure (6.3.10): Pipelined integer phase multiplier computing the k-tuple of a phase sequence, $\phi(n)$, modulo 2^M.

6.3.6 Synthesising Partial Phase Sequences

In this section we present an enhancement to the HAS processing architectures illustrated in Figures (6.3.7) and (6.3.9), which generate partials that take on *fractional* multiples of the fundamental frequency. Moreover, partial frequency is now continuously time-variable at the sample rate according to a distinct control parameter and baselined to an harmonic multiple of the fundamental. We may now implement the PAS processing model where partials are not constrained to follow an harmonic distribution and where we assume that the partial start-phase parameter is constant (i.e. time invariant) and denoted by Φ_j.

We begin by considering a phase accumulator whose phase increment is obtained from a pitch-to-phase increment translation table as illustrated in Figure (6.3.4). Hence, we have $f_0 = \frac{\varphi f_s}{2^M} = B\gamma^{\rho}$ and so we obtain $\varphi = \frac{B\gamma^{\rho} 2^M}{f_s} \in [0, 2^{M-1} - 1]$. We now define $\varphi'(n)$ as the time-varying phase increment corresponding to a time-varying pitch offset, $\beta(n)$, applied to $\rho(n)$, thus $\varphi'(n) = \frac{B\gamma^{\rho(n)+\beta(n)} 2^M}{f_s} \in [0, 2^{M-1} - 1]$. It is evident that as defined by Eq. (6.3.2), $\varphi(n)$ and $\varphi'(n)$ are strictly *integer* quantities and we observe that the *rounding quantiser function* $z \to \lfloor z + 0.5 \rfloor$, $z \in \Re$ as applied in Eq. (6.3.2) is non-linear. Hence, for two arbitrary real numbers x and y we have $\lfloor (x \pm y) + 0.5 \rfloor \neq \lfloor x + 0.5 \rfloor \pm \lfloor y + 0.5 \rfloor$ and observe that the error magnitude is bound according to $\left| \lfloor (x \pm y) + 0.5 \rfloor - (\lfloor x + 0.5 \rfloor \pm \lfloor y + 0.5 \rfloor) \right| \in [0,1]$ and strictly only takes on values of 0 or 1. In the development of Eqs. (6.3.8) and (6.3.9) we reason that using rounded $\varphi(n)$ and $\varphi'(n)$ values (as required for a fixed-point hardware implementation) instead of full-precision fractional values, introduces an error

magnitude of no more than unity and so the resulting frequency error magnitude is bound by $\frac{f_s}{2^M} \approx 0$ (i.e. one part in 2^M). Hence, as written here $\varphi(n)$ and $\varphi'(n)$ are fractional quantities and we assume for our present discussion that they are full-precision fractional quantities. We define the phase sequences which correspond to these phase increments, thus:

$$\begin{aligned}
\phi(n) &= \left\langle TB2^M \sum_{m=1}^{n} \gamma^{\rho(m)} \right\rangle_{2^M} \\
\phi'(n) &= \left\langle TB2^M \sum_{m=1}^{n} \gamma^{\rho(m)+\beta(m)} \right\rangle_{2^M} \\
k\phi(n) &= \left\langle kTB2^M \sum_{m=1}^{n} \gamma^{\rho(m)} \right\rangle_{2^M}
\end{aligned} \tag{6.3.8}$$

and a new phase sequence, $\mu(n)$, thus:

$$\begin{aligned}
\mu(n) &= \left(\phi'(n) - \phi(n) + k\phi(n)\right) \\
&= \left(\phi'(n) + (k-1)\phi(n)\right) \\
&= \left\langle TB2^M \left(\sum_{m=1}^{n} \gamma^{\rho(m)+\beta(m)} + k\sum_{m=1}^{n} \gamma^{\rho(m)} - \sum_{m=1}^{n} \gamma^{\rho(m)} \right) \right\rangle_{2^M} \\
&= \left\langle TB2^M \sum_{m=1}^{n} \gamma^{\rho(m)} \left(\gamma^{\beta(m)} + k - 1\right) \right\rangle_{2^M}
\end{aligned} \tag{6.3.9}$$

The idealised (i.e. $L = 2^M$) sinusoidal phase-amplitude mapping lookup table is defined by $\mathrm{S}[a] = \cos\left(2\pi \frac{a}{2^M}\right)$ and so with $y(n) = \mathrm{S}[\mu(n)]$ and ignoring any start-phase offset, we obtain:

$$y(n) = \cos\left(2\pi TB \sum_{m=1}^{n} \gamma^{\rho(m)} \left(\gamma^{\beta(m)} + k - 1\right)\right) \tag{6.3.10}$$

Comparing Eq. (6.3.10) with the DT sinusoid $y(n) = \cos\left(2\pi T \sum_{m=1}^{n} f_0(m)\right)$ (see Eq. (2.3.4)) where $f_0(n)$ denotes the instantaneous frequency, we deduce by comparing terms that:

$$f_0(n) = B\gamma^{\rho(n)}\left(\gamma^{\beta(n)} + k - 1\right) \qquad (6.3.11)$$

Eq. (6.3.9) therefore describes a phase sequence whose frequency is controlled by $\rho(n)$ with a *multiplying* factor, $\left(\gamma^{\beta(n)} + k - 1\right)$, allowing *independent* control of integer harmonic multiplier, k, and fractional component, $\left(\gamma^{\beta(n)} - 1\right)$.

The advantage of this representation is evident when we apply it to the PAS model over N_p partials where the β and k terms take on j subscripts denoting the j^{th} partial parameter. We now have *independent* time-varying control of fundamental frequency (pitch) through $\rho(n)$, j^{th} partial harmonic multiplier through k_j, j^{th} partial "fine tuning" through $\beta_j(n)$, in addition to the j^{th} partial start-phase and amplitude parameters Φ_j and $A_j(n)$, respectively. This model enables application of fine-grained partial frequency envelopes via the $\beta_j(n)$ parameter in line with the PAS model. The j^{th} partial frequency is specified as a fractional multiple of the fundamental comprising an integer "harmonic multiplier" component k_j and a "fractional multiplier" component $\left(\gamma^{\beta_j(n)} - 1\right)$. For example, we may now specify the "3.275$^{\text{th}}$ harmonic" independently of the fundamental frequency. Moreover, since $\beta_j(n)$ is time-varying, we may specify a given partial as an arbitrary time-varying fractional multiple of the fundamental through piecewise-linear variation of $\beta_j(n)$. Figure (6.3.11) illustrates the hardware architecture for this processing model, where we depict the fine-tuning (i.e. $\beta_j(n)$) signal path in

dashed lines. This model computes the weighted sum of N_p partials with time-varying parameterisation in $N_p = N_c$ time-slots, thus:

$$y(n) = \sum_{j=0}^{N_p-1} A_j(n-2)\cos\left(\phi_j(n)\right)$$

$$\phi_j(n) = 2\pi TB\sum_{m=1}^{n}\left(\gamma^{\rho(m-2)}\left(\gamma^{\beta_j(m-3)} + k_j - 1\right)\right) + \Phi_j \tag{6.3.12}$$

Sample index terms within Eq. (6.3.12) reflect the respective pipeline delays for the three time-varying parameters $A_j(n)$, $\rho(n)$ and $\beta_j(n)$ within the arithmetic model of Figure (6.3.11).

The PAS processing architecture of Figure (6.3.11) enables the amplitude, frequency and start-phase of individual partials to be independently controlled in real-time. Partial frequency control is effected with only addition and table-lookup operations performed to M-bit precision. The hardware imposition compared to the HAS processing architecture of Figure (6.3.9) comprises an M-bit phase accumulator, φ lookup table, four adders and two DPM blocks to store the $\beta_j(n)$ parameters and state-variables. It is evident from Eq. (6.3.9) that the phase subtractor is unnecessary if $(k_j - 1)$ rather than k_j values are stored in the partial k DPM. Metaparameterisation of partial frequency relative to the fundamental as outlined in section (2.6.5) is now imposed by an *additive* offset to the $\beta_j(n)$ parameter and incurs less computation burden than multiplicative scaling of the phase increment for each partial in the metaparameter set.

Finally, we consider a modification to the partial fractional multiplier model which provides a *frequency offset* to each partial according to the *absolute* frequency offset parameter $\beta_j(n)$. We first redefine the $\phi'(n)$ term in Eqs. (6.3.8) so that

$\phi'(n) = \left\langle TB2^M \sum_{m=1}^{n} \left(B\gamma^{\rho(m)} + \beta(m)\right)\right\rangle_{2^M}$ where $\beta(n)$ represents a time-varying frequency offset in Hz. Following similar reasoning to that applied in the development of Eq. (6.3.9) we obtain:

$$\mu(n) = \left\langle kTB2^M \sum_{m=1}^{n} \left(\gamma^{\rho(m)} + \beta(m)\right)\right\rangle_{2^M} \qquad (6.3.13)$$

The idealised (i.e. $L = 2^M$) sinusoidal phase-amplitude mapping lookup table is defined by $\mathbf{S}[a] = \cos\left(2\pi \frac{a}{2^M}\right)$ and so with $y(n) = \mathbf{S}[\mu(n)]$, we obtain:

$$y(n) = \cos\left(2\pi kTB \sum_{m=1}^{n} \left(\gamma^{\rho(m)} + \beta(m)\right)\right) \qquad (6.3.14)$$

Comparing Eq. (6.3.14) with the DT sinusoid $y(n) = \cos\left(2\pi T \sum_{m=1}^{n} f_0(m)\right)$, we obtain:

$$f_0(n) = kB\gamma^{\rho(n)} + \beta(n) \qquad (6.3.15)$$

Eq. (6.3.13) therefore describes a phase sequence whose fundamental frequency is controlled by $\rho(n)$ with *independent* control of integer harmonic multiplier k, and frequency offset $\beta(n)$, in Hz. Figure (6.3.16) illustrates the modified section of the pipelined multiple-voice processing model depicted in Figure (6.3.13a) to effect partial frequency offset according to the $\beta_j(n)$ parameter which now denotes an absolute frequency offset to the j^{th} partial. This modified model computes the weighted sum of N_p partials in $N_p = N_c$ time-slots executing the PAS model, thus:

$$y(n) = \sum_{j=0}^{N_p-1} A_j(n-2)\cos\left(\phi_j(n)\right)$$

$$\phi_j(n) = 2\pi k_j TB \sum_{m=1}^{n} \left(\gamma^{\rho(m-2)} + \beta_j(m-3)\right) + \Phi_j \qquad (6.3.16)$$

$$y(n) = \sum_{j=0}^{N_p-1} A_j(n-2)\cos\left[\phi_j(n)\right]$$

$$\phi_j(n) = 2\pi TB \sum_{m=1}^{n}\left[\gamma^{\rho(m-2)}\left[\gamma^{\beta_j(m-3)} + k_j - 1\right]\right] + \Phi_j$$

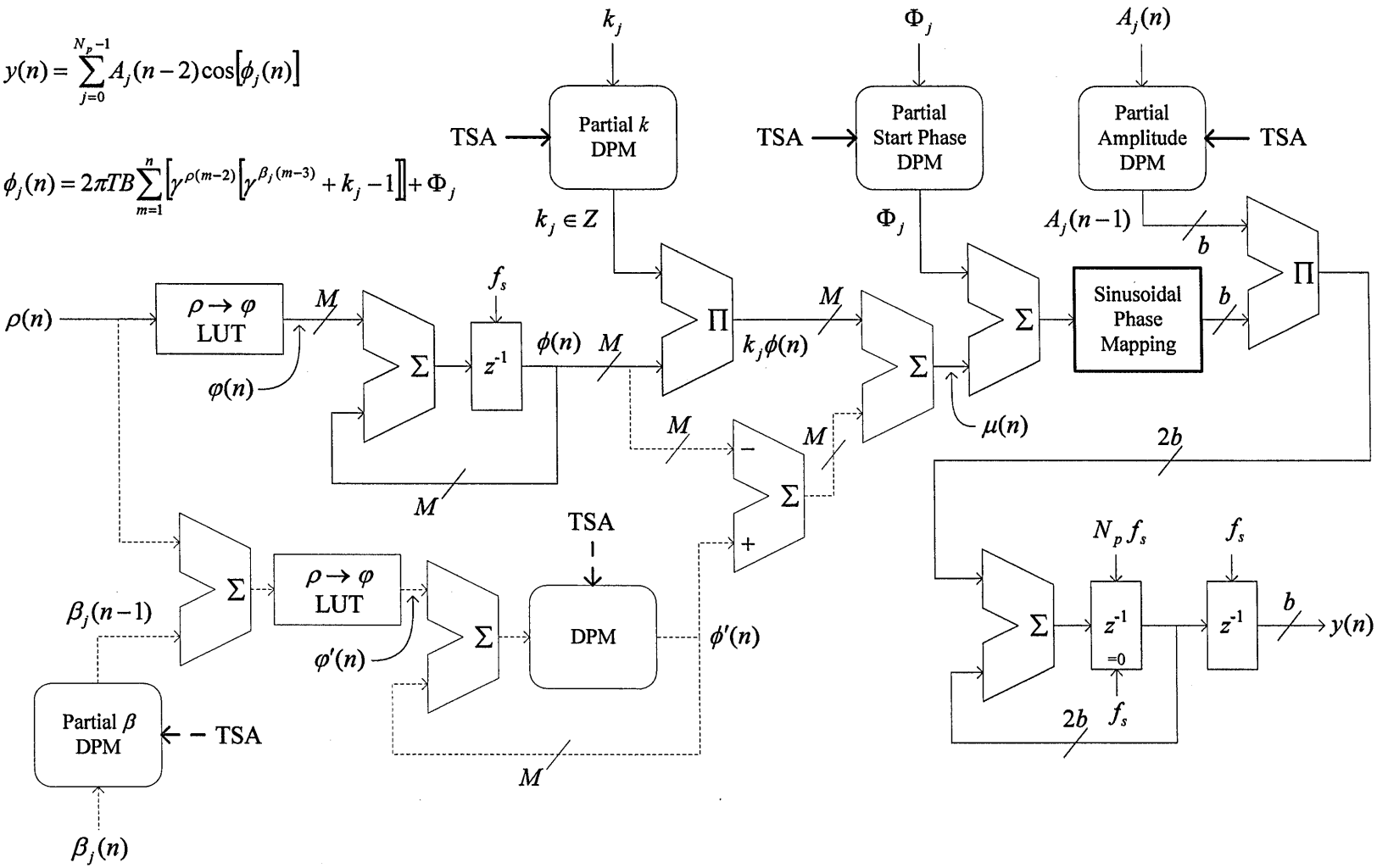


Figure (6.3.11): Implementation of the PAS arithmetic process model using phase domain processing. This architecture depicts a ***single voice*** *synthesiser linearly combining N_p partials with arbitrary pseudo-harmonic frequencies according to k_j and $\beta_j(n)$.*

6.3.7 A Multiple Voice PAS Processing Architecture

The PAS processing architecture of Figure (6.3.11) can be extended to synthesise multiple *voices* by block pipelining the fundamental phase accumulator computations. Denoting the number of voices to be generated by υ where each voice comprises N_p partials, we observe that DPM blocks of length υ are now required for each of the $\rho(n)$ and $\phi(n)$ variables. All other parameter and state-variable DPMs now require υN_p locations reflecting the υN_p partials computed across υ voices. It is conceivable that within a completely generic architecture, partial allocation across voices is non-homogeneous and dynamically reconfigurable enabling optimal allocation of partials to those voices that require a rich partial composition. For a given arithmetic architecture and processing speed, we require the υN_p product to be constant since this determines the total number of arithmetic operations required within each sample period.

Assuming a homogeneous allocation of partials to voices, the sample period is sub-divided into υ distinct voice computation time slots by the *most significant bits* (MSBs) of the TSA variable. Each of the voice computation slots is further sub-divided into N_p partial computation time slots by the *least significant bits* (LSBs) of the TSA variable. Hence, the TSA now addresses υ sets of N_p partials – one unique set for each voice. The MSBs of the TSA address the fundamental phase accumulator DPMs (i.e. over υ locations) with the LSBs addressing the partial parameter and state variable DPMs (i.e. over N_p locations). We now have three distinct processing levels defined by processing clock speed. At the control level we have sample rate processing of the time-varying PAS parameters, $\rho_i(n)$, $\beta_j(n)$ and $A_j(n)$ which are fed into their respective DPMs. At the voice level we compute the fundamental phase sequences for each voice at υf_s

operations per second. Finally, at the partial level we compute the individual partials for each voice and sum the results into a composite value at vN_pf_s operations per second.

Figure (6.3.12) illustrates the multiple-voice PAS processing model where the new DPM blocks are shown shaded. It is evident that this architecture may be pipelined at both the block and elemental processing levels to increase throughput. Figure (6.3.13a) illustrates the pipelined form of the architecture presented in Figure (6.3.12) where sub-sample pipeline registers separate each processing stage and form a twelve stage pipelined processor. Figure (6.3.13b) illustrates the pipelined arithmetic processing model for the sinusoidal phase-amplitude mapping block shown in Figure (6.3.13a). PAS parameters excluding $\rho_i(n)$ are delayed through a cascade of pipeline registers to compensate for time-skew inherent in the multi-parameter pipeline. Figure (6.3.14) illustrates the two DPM addressing models corresponding to Figures (6.3.12) and (6.3.13a). Figure (6.3.15) illustrates a hierarchical timing diagram of the pipelined arithmetic processing taking place in the processing model of Figure (6.3.13a). The timing diagram illustrates staggered computation of the first partial within the first voice as it propagates along the twelve-stage pipeline. Key computation points are denoted by circled numerical references which correspond with those shown in Figure (6.3.13a). The twelve-clock latency of this fast pipeline stage must be accounted for by a corresponding delay in the final $y(n)$ register clock. Overall latency is three sample periods comprising the two consecutive DPM elements and the final output register.

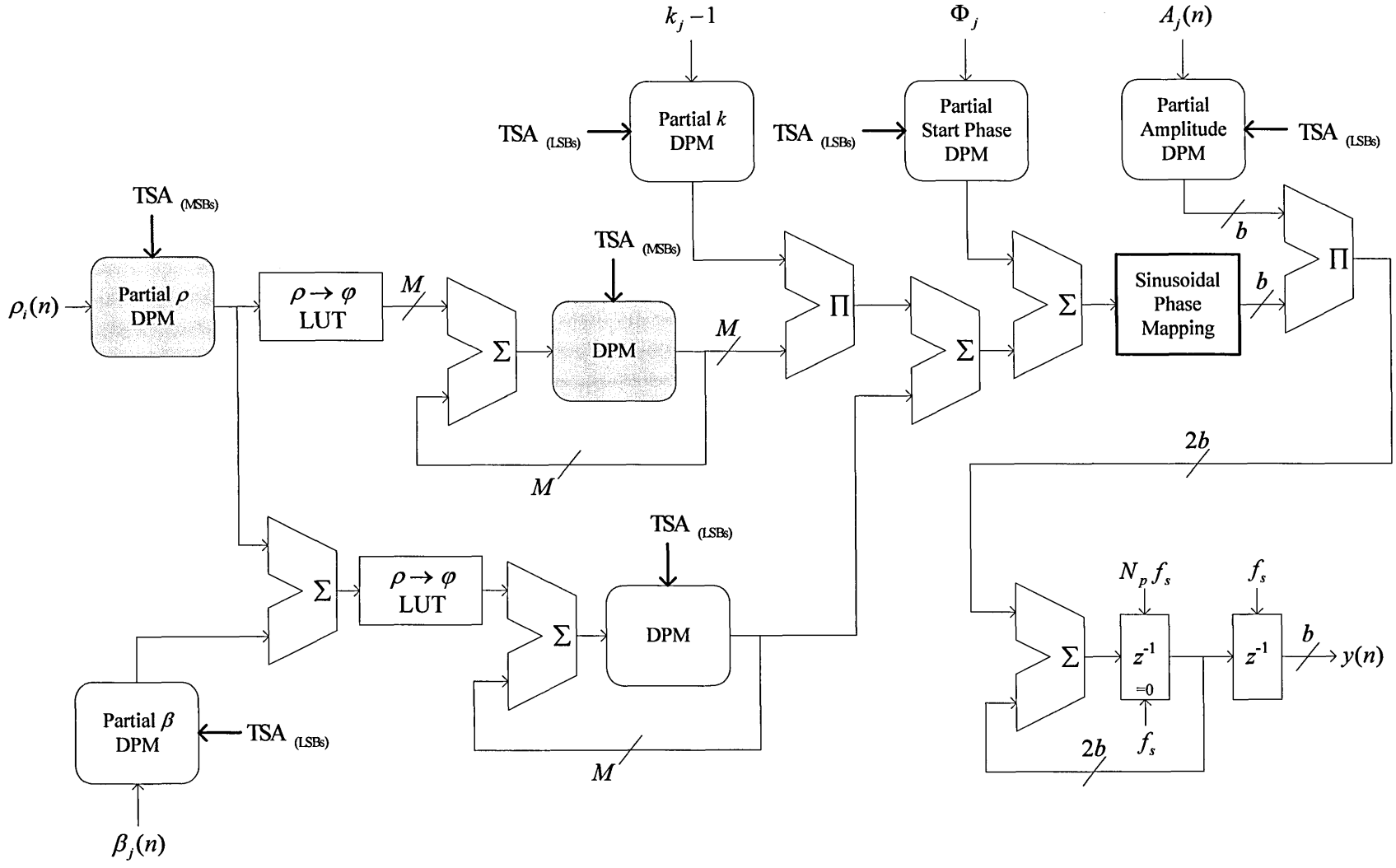


Figure (6.3.12): Implementation of the PAS arithmetic process model using phase domain processing. This architecture depicts a ***multiple-voice*** *synthesiser, each voice linearly combining N_p partials with arbitrary, pseudo-harmonic frequencies according to k_j and $\beta_j(n)$.*

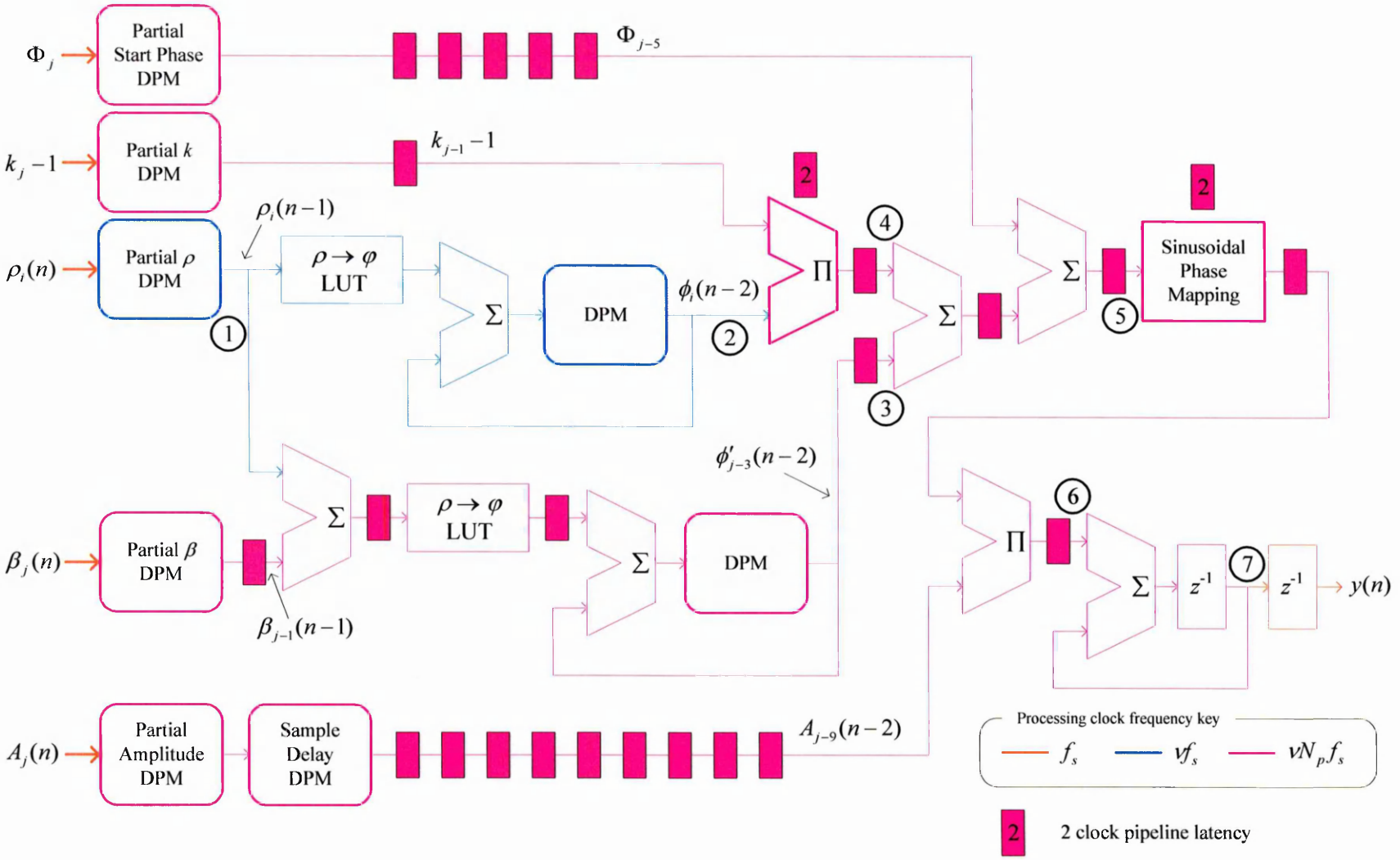


Figure (6.3.13a): Pipelined processing model of the multiple-voice PAS algorithm. This processor computes ν voices each comprising N_p partials with time-varying frequencies according to k_j and β_j. Colour codes depict the three levels of processing rate.

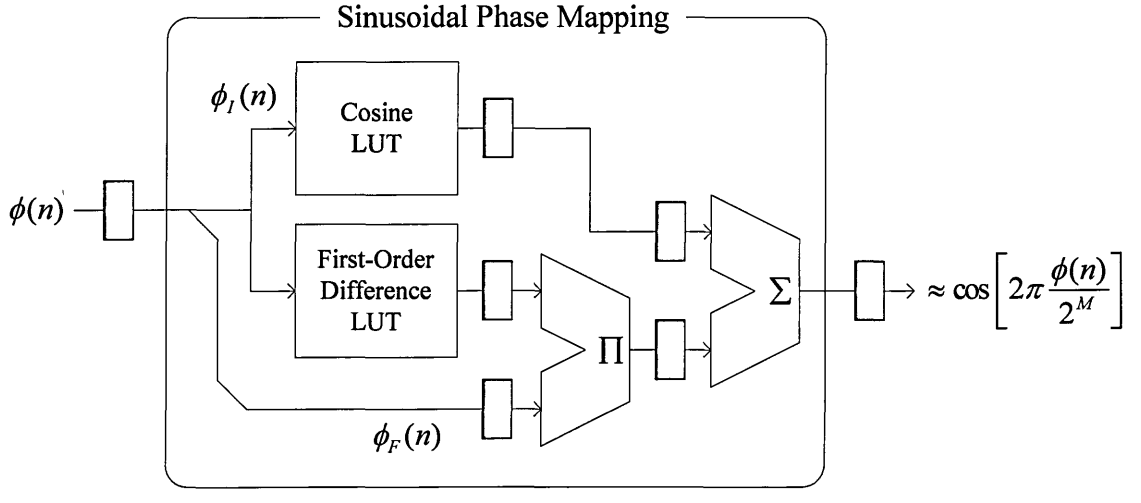


Figure (6.3.13b): Pipelined processing model of the sinusoidal phase-amplitude mapping block used in Figure (6.3.13a). This model represents the pipelined form of Figure (6.3.8).

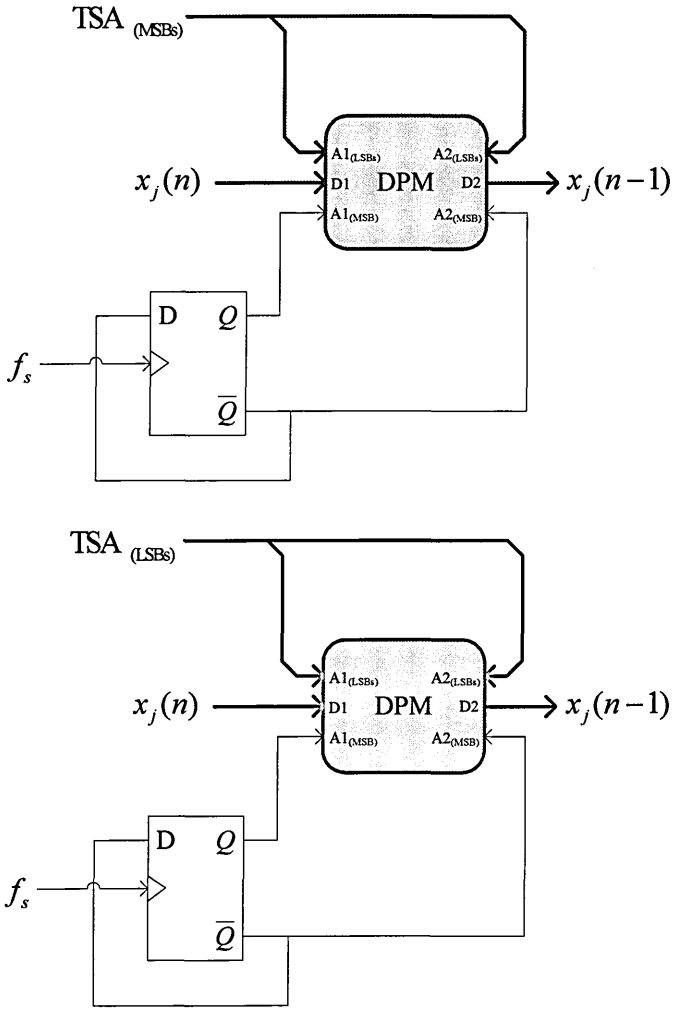


Figure (6.3.14): Dual port memory (DPM) addressing for the multiple-voice pipelined PAS processing models of Figures (6.3.12) and (6.3.13a).

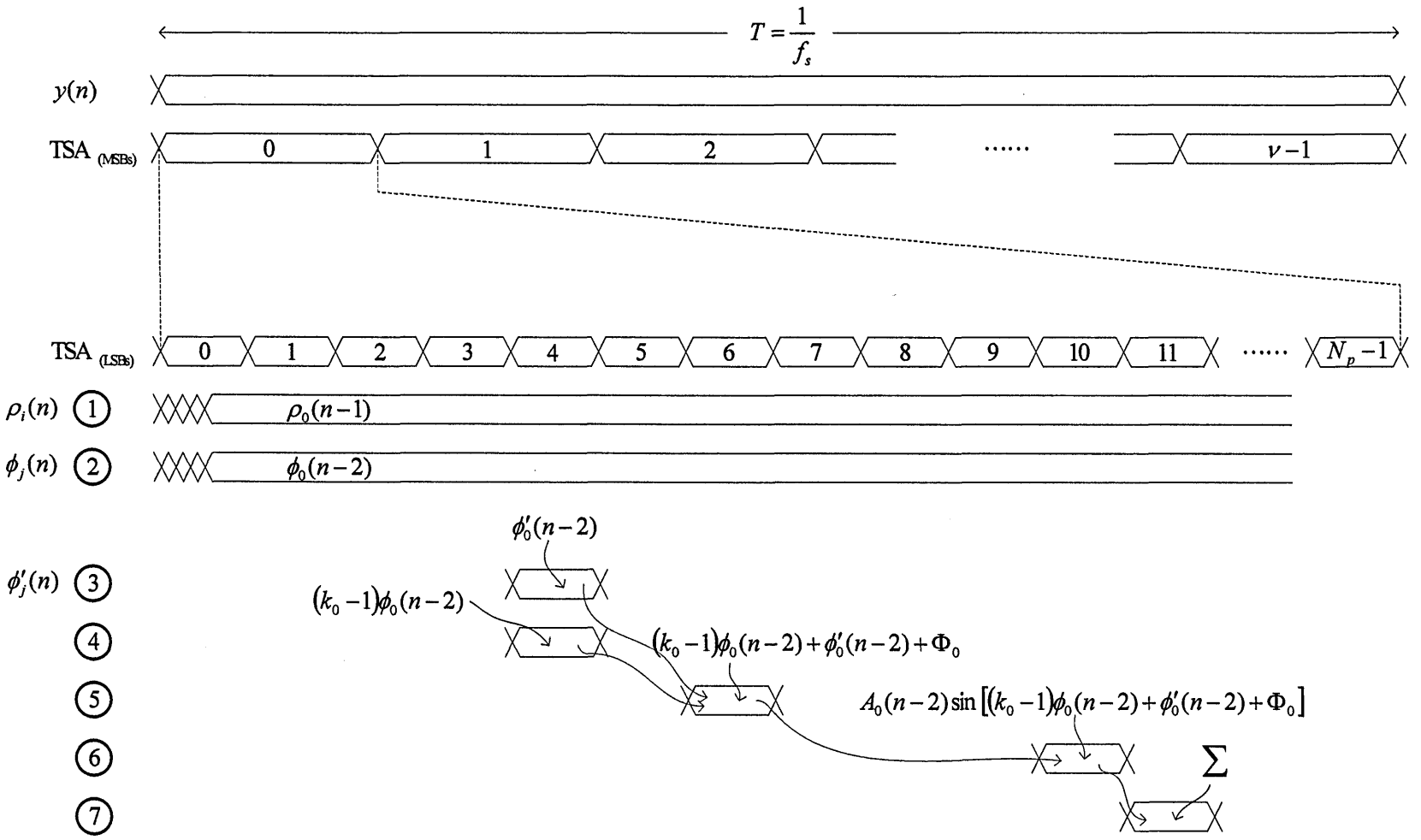


Figure (6.3.15): Simplified timing diagram of the pipelined processing model shown in Figure (6.3.13). Computation of the first partial of the first voice is seen propagating down the pipe at key computation stages depicted by the numbered references shown in Figure (6.3.13).

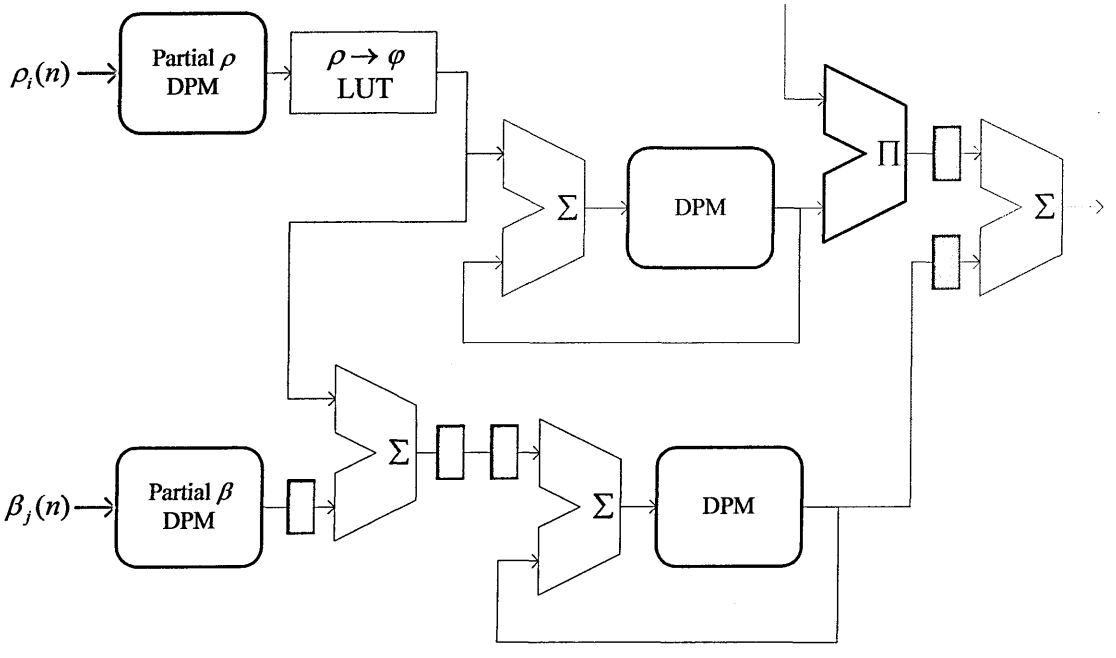


Figure (6.3.16): Modification to the processing model of Figure (6.3.13a) to effect partial frequency offset according to the $\beta_j(n)$ parameter. (Note the residual pipeline registers to ensure correct timing skew correction.)

6.3.8 Simulation Results

In this section we present simulation results for the partial *fractional multiplier* and *frequency offset* phase domain PAS processing models presented in section (6.3.6) using the corresponding Mathcad PAS models presented in Appendix B. Figure (6.3.17) illustrates an example waveform sequence whose constituent partials have an inharmonic frequency distribution causing a characteristic time-varying waveshape. Partial fractional multiplier values vary pseudo-randomly with a maximum value of ≈1.059 (1 semitone) over four partials whose amplitudes follow a -3 dB/octave roll-off slope. Figure (6.3.18) illustrates SNR behaviour over 200 pseudo-random partial fractional multiplier distributions for three values of spectrum roll-off slope with $N_p = 64$, $N = 1$ and $I = 10$. Each point corresponds to a distinct set of N_p pseudo-random β_k values with a maximum fractional multiplier value of 1.25. We observe essentially invariant SNR with partial fractional multiplier distribution whose average value is consistent with the linearly interpolated phase-amplitude mapping used.

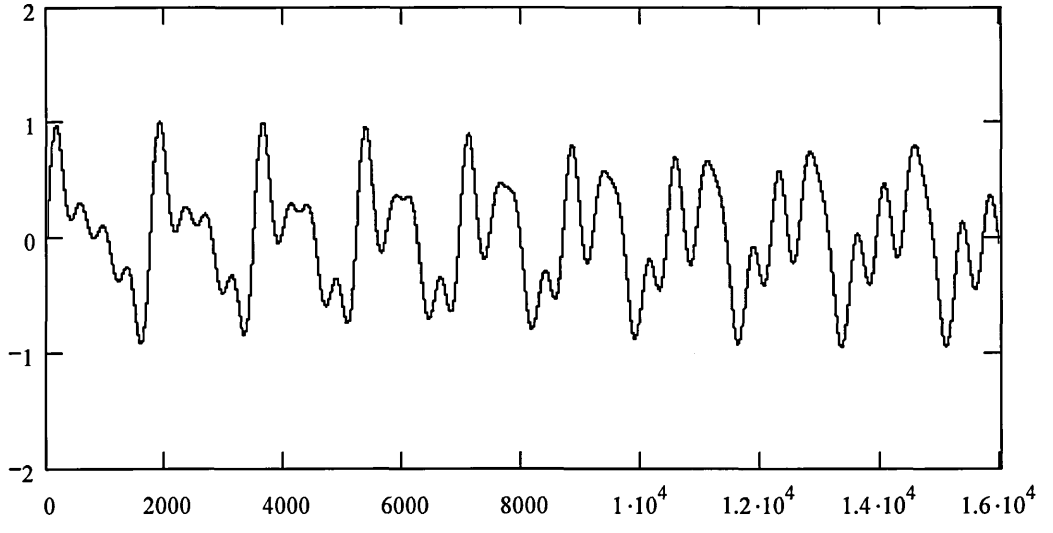


Figure (6.3.17): Example waveform synthesised using the PAS processing model. Notice the evolving waveshape as the inharmonic partials beat in frequency.

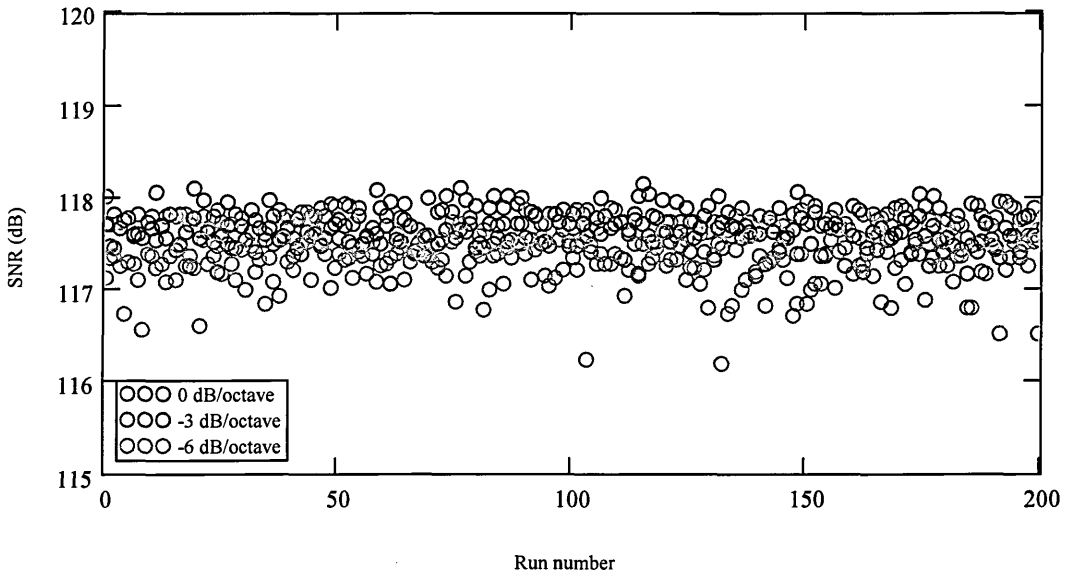


Figure (6.3.18): SNR variation over 200 pseudo-random partial fractional multiplier distributions with three spectrum roll-off slopes and full-precision arithmetic.

Figures (6.3.19) and (6.3.20) illustrate time-varying spectra for the partial fractional multiplier and frequency offset models using Mathcad models given in Appendix B. The $\beta_j(n)$ parameters for the second, third, fifth and eighth partials are arranged to take on triangular PWL envelopes. Fundamental frequency is set at 55 Hz (A1) and partials have a spectral envelope of -6 dB/octave in both cases. Figure (6.3.19) sets the partial fractional multiplier envelope amplitudes to 0.2, 0.3, 0.5 and 0.8 for the second, third, fifth and eighth partials, respectively[8]. Figure (6.3.20) sets the partial frequency offset envelope amplitudes to 20, 30, 50 and -80 Hz for the second, third, fifth and eighth partials, respectively. Both simulations show 64 spectra visualised as contour and surface plots where the time-varying partial frequency envelopes are clearly visible, precisely in line with the underlying PWL $\beta_j(n)$ envelope.

[8] For example, the third partial varies from the harmonic multiple of 3 to 3.3 times the fundamental frequency over the PWL envelope.

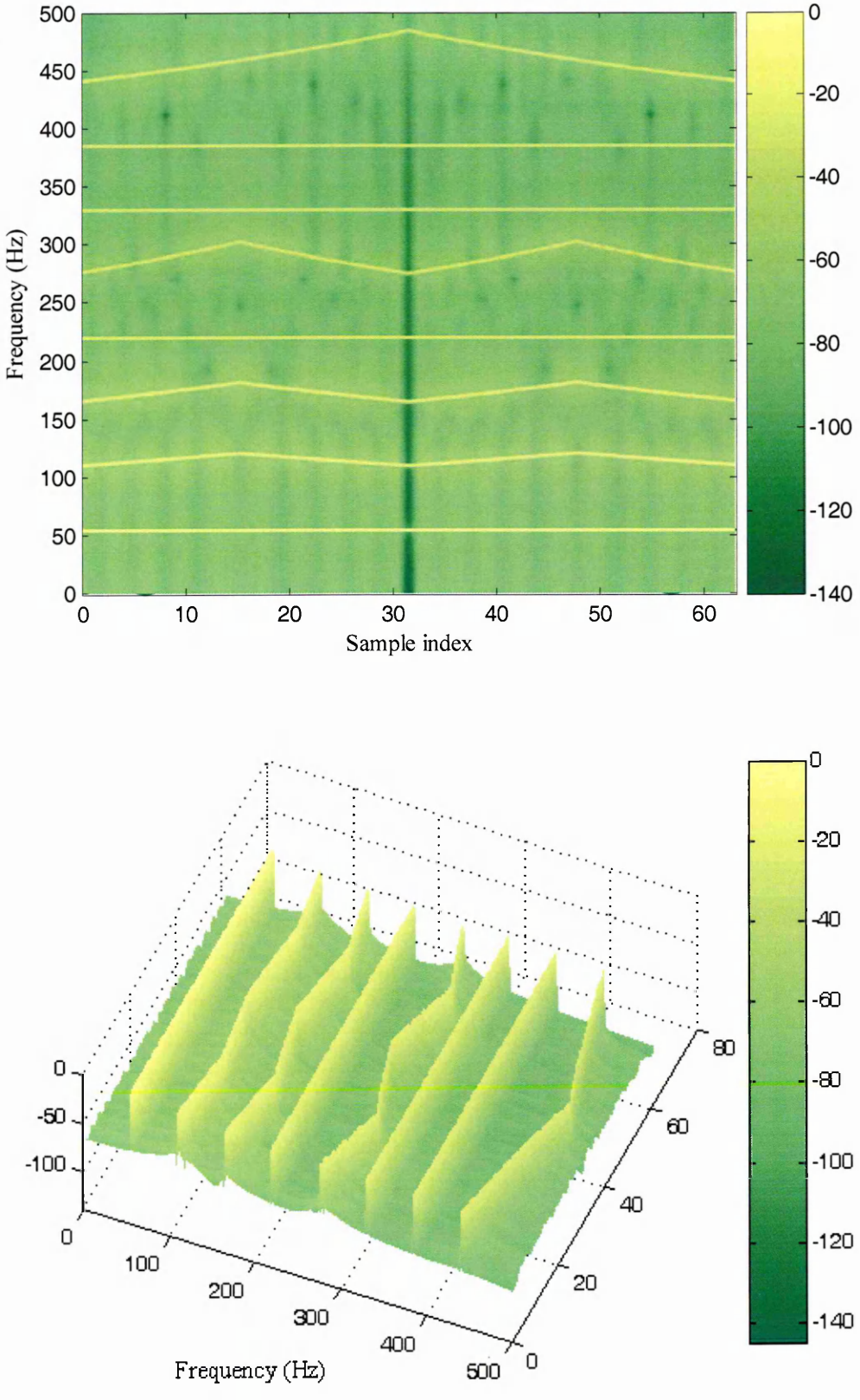


Figure (6.3.19): Spectrograms for the partial fractional multiplier model.

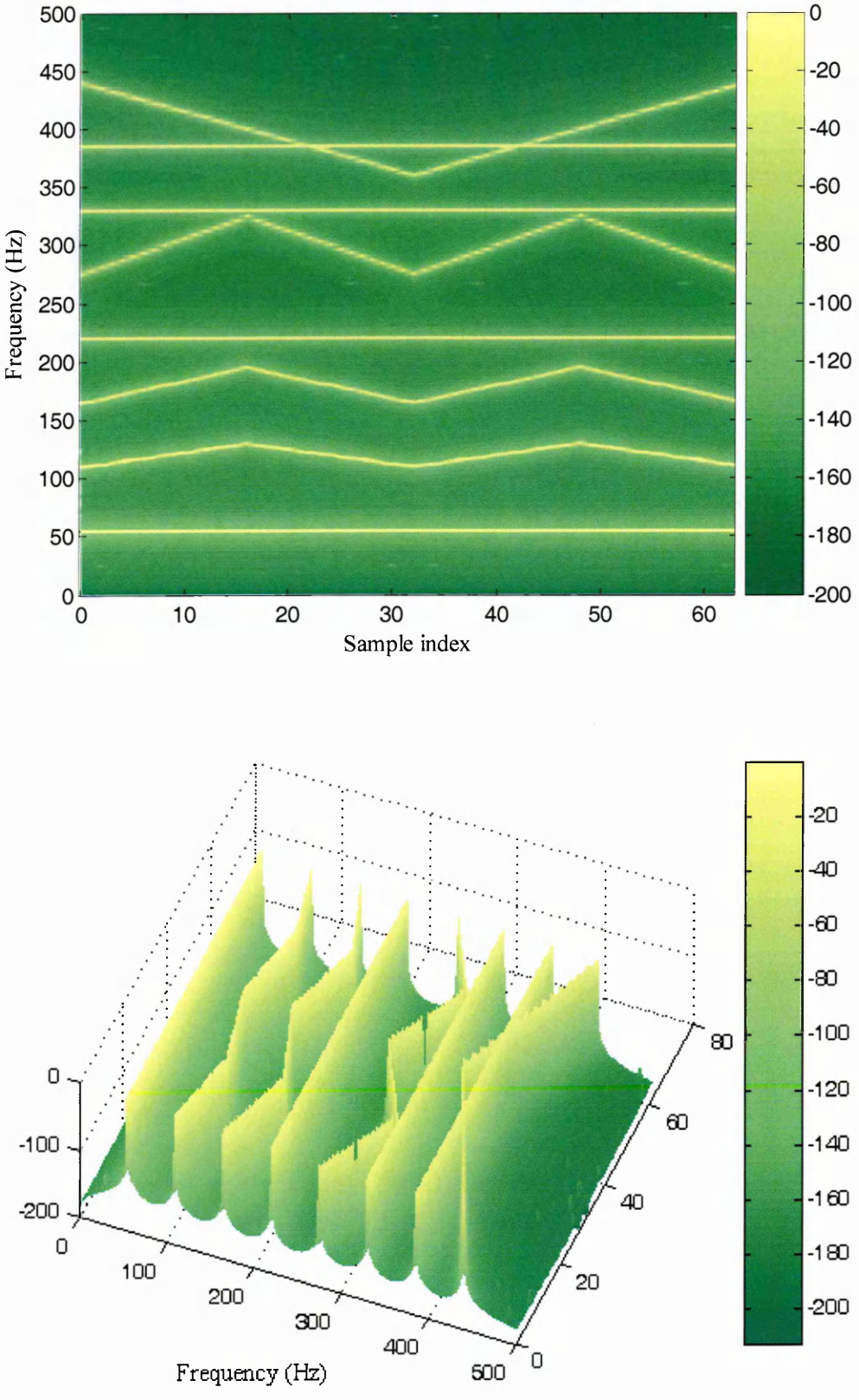


Figure (6.3.20): Spectrograms for the partial frequency offset model.

6.4 Conclusions

In this chapter we have presented arithmetic processing architectures which underpin both the WLS and PAS paradigms. The consecutive access wavetable memory appears unique in computer music applications and represents an original contribution from this research that enables significant throughput enhancement in multi-voice interpolated WLS compared to multiple accesses of a single wavetable memory. This data-parallel vector memory architecture and the interpolation processing which proceeds it, are inherently "pipeline friendly" and may be readily extended to effect *wavetable interpolation* which implements the SIS model as presented in section (6.2.5). We have investigated the application of lookup tables to reduce interpolation coefficient arithmetic overhead and extended this to include the sample reordering function. However, utility of the WLS model is fundamentally constrained by the inherently fixed (i.e. pre-computed) wavetable spectral characteristics which limit the highest synthesised frequency according to the onset of upper harmonic aliasing. Assigning multiple wavetables to each voice with progressively reducing upper harmonic content and selected according to fundamental frequency circumvents the aliasing problem. This method increases memory overhead in accordance with the multiplicity of wavetables, although the falling cost-to-capacity ratio of semiconductor memory at this point in history affords compensation. However, wavetable "fill time" increases in direct proportion to the number of wavetables with this approach.

Building on the concept of phase domain processing introduced in Chapter 4 and the arithmetic structure first proposed by Chamberlin [1976], we have investigated an arithmetic processing architecture which executes the HAS model in real-time. We have proceeded to show how this architecture is extendable to include the generic PAS model providing independent, time-varying control of partial amplitude, frequency and start-

phase. A refinement to the PAS model has been presented which provides time-varying partial *frequency offset*.

Having control granularity at the individual partial level allows upper harmonic aliasing to be efficiently circumvented by truncating the linearly combined partial series as a function of fundamental frequency (i.e. partial frequencies which would lie above the Nyquist limit have amplitude set to zero). The control parameter interface to this architecture appears as memory and may therefore be mapped into a host computer memory space. Parametric data control is supported at the system sample rate as required for optimal PAS control as discussed in Chapter 2. However, in a real embodiment the partial amplitude and frequency parameter DPM blocks will be fed by a dedicated PWL envelope processor whose segment slope and breakpoint data sets are supplied from the host computer. The PWL envelope processor architecture is envisaged as a block pipeline which presents a DPM interface "image" to the host processor. This architecture may be implemented with current FPGA and multi-port memory technologies. Pipelined multiplier, logic and memory speeds above 200 MHz are currently reported for the Xilinx Virtex II FPGAs [Xilinx, 2004] and Integrated Device Technology synchronous DPMs [Smith, 2000]. As an indication, a sample rate of 48 kHz and a 5 ns computation cycle time enables a 64 voice synthesiser with each voice linearly combining up to 64 partials or a total of 4096 partials.

We have presented simulated SNR behaviour for the phase domain PAS processing model which confirms expected performance given the linearly interpolated sinusoidal phase mapping over a range of pseudo-random, inharmonic partial frequency multiples. We conclude by presenting simulated time-varying spectra for the partial *fractional multiplier* and *frequency offset* models, which clearly illustrate the time-varying partial frequency envelope according to an underlying PWL control function.

Chapter 7 Conclusions

7.1 Introduction

This thesis has presented a structured investigation of the two underpinning hypotheses given in section (1.3.2). A critical review of prevalent synthesis techniques reported in the literature and presented in Chapter 2 has set additive synthesis in context ahead of further investigation and development in later chapters. Moreover, this review has substantiated the motivating assumption that sinusoidal additive synthesis provides complete accessibility to the elemental parts of timbral composition, in line with models of human timbral perception.

Our review of the HAS and PAS mathematical foundations in sections (2.3.2) and (2.6.2) reveals the inherent concordance with phase-accumulating frequency synthesis and distinct phase-amplitude mapping. The difference equation definition of an oscillatory phase sequence given by Eq. (2.6.2) encapsulates this view. We have shown that the phase-accumulating model provides flexibility, supporting time-invariant implementation of the HAS model with fixed phase-amplitude mapping wavetables computed "off-line" according to Eq. (4.1.3), or *direct* implementation with time-varying parameters according to Eq. (2.3.6) and developed in section (6.3). Extending the phase-accumulating model to include the PAS model given by Eq. (2.3.5) and developed in section (6.3.6), enables the SMS model presented in section (2.3.3) and expressed by Eq. (2.3.7) to be effected. Software implementations of hitherto IFFT based SMS models require a "spectrally shaped" noise model whose implementation we have not considered due to the computational simplicity of filtered noise synthesis according to the subtractive synthesis model discussed in section (2.3.4).

7.2 Research Objectives

Our critical review of the two principal sinusoidal oscillator techniques presented in Chapter 3 confirms the efficacy of phase-accumulated over recursive algorithms. A detailed development of initial condition values in the second-order direct-form recursive oscillator that provides phase-continuous, amplitude-invariant frequency transition, reveals significant computational imposition compared to the phase-accumulating case which is inherently phase-continuous. The compelling arguments for the phase-accumulating oscillator may be summarised thus:

- Linear time-varying frequency control which can be updated at the sample rate, with no initial condition imposition;
- Constant frequency control resolution which is a simple function of accumulator width and sample rate alone;
- Inherently phase-continuous frequency transition and time-invariant oscillation amplitude;
- Simple arithmetic architecture requiring only one state-variable;
- Distinct phase-amplitude mapping using interpolated wavetable lookup is conducive to a pipelined arithmetic architecture;
- Supports phase-domain processing prior to phase-amplitude mapping and thereby the generation of consecutive and non-consecutive harmonic sinusoid sequences.

The distinct phase-amplitude mapping operation is an advantage since it enables synthesis of non-sinusoidal waveforms via appropriate phase-mapping wavetable definition. Linearly interpolated sinusoidal phase-amplitude mapping is effected with pipelined lookup tables and interpolation arithmetic having optimised word sizes and does not detract from computational throughput.

Chapter 3 introduced the wavetable as a fundamental element in the phase-amplitude mapping process, proceeding to review the concept of "sampling a tabulated signal" which we generalise as *wavetable lookup synthesis* (WLS) in Chapter 4. We have investigated frequency control in the phase accumulating oscillator and defined the design parameters for frequency control *resolution* compatible with computer music synthesis requirements based on reported pitch perception thresholds. Section (4.2.3) develops the inherent phase-continuous nature of the phase-accumulation process in contrast to the direct-form recursive algorithm presented in Chapter 3. For completeness, section (4.3) discusses the "sample rate conversion" view of WLS which is pertinent to the *sampling synthesis* subclass reviewed in Chapter 2 and can be considered as a special case of the WLS model. Furthermore, section (4.3.5) reviews the relationship between pitch shift and phase fraction field width peculiar to the phase-accumulating sampling synthesis model.

Chapter 5 introduces the amplitude error mechanism consequential to phase truncation or rounding and proceeds to investigate interpolated WLS as a means of reducing the magnitude of these errors. Interpolation of tabulated data according to a fractional address is reviewed and arithmetic overhead for various interpolation orders is discussed. To objectively assess interpolation effectiveness we introduce the SNR and error spectrum metrics and present a comparative assessment of different interpolated WLS scenarios in section (5.5) based on numerical models presented in Appendix B with sinusoidal and multi-harmonic wavetables. A range of multi-harmonic "test" wavetables have been used in these simulations whose tabulated waveform spectra are based upon piecewise-linear approximations of the long term average spectra of various music types reported in the literature. We assume that these "test spectra" represent a reasonable approximation to worst case instrument spectra in a synthesis environment.

The results confirm that linear interpolation provides SNR performance comparable with 16-bit SQNR for sinusoidal wavetables of length 512 words. TIPM is shown to provide SNR bound by quantisation noise as predicted and offers maximum utility when quadrature signals with optimal (i.e. $\frac{2\pi}{2^M}$ radian) phase control resolution is required. Furthermore, it is evident that quadrature oscillators with ultra-precise phase and frequency control as afforded by the TIPM model, have utility in the implementation of advanced lock-in amplifiers within the field of instrumentation and signal recovery [Meade, 1982].

Cubic interpolation provides SNR performance comparable with 16-bit SQNR using wavetables of length 32k words which tabulate the worst case (i.e. richest) multi-harmonic spectrum of those simulated. We present simulated contour plots which illustrate the two-dimensional behaviour of SNR as a function of *wavetable length* and *spectrum roll-off slope* for the four interpolation orders investigated. These plots provide a concise summary of SNR performance against variation of two principal parameters in multi-harmonic WLS and are of utility in estimating wavetable length and interpolation order for a given synthesis application.

WLS implements the HAS model but is always constrained to have partials which are *harmonic* multiples of the fundamental. Dynamic timbral evolution is effected by sequencing manifold wavetables whose pre-computed fixed spectral content represent distinct points in "timbre space" analogous to frames of a film. The SIS model provides linear interpolation between fixed wavetables in a sequence and thereby finer timbral granularity with no increase in wavetable memory imposition. MWS is an alternative additive synthesis model which linearly combines non-sinusoidal basis components stored in distinct wavetables. Despite the performance of cubically interpolated WLS, three problems remain for all embodiments of the WLS model – aliasing of upper

harmonics when $N_h\varphi > 2^{M-1}$, wavetable computation according to Eq. (4.1.3) which must be executed for each wavetable in the set and wavetable loading time from a mass-storage device.

Chapter 6 considers arithmetic processing architectures for the WLS, HAS and PAS models. Section (6.2) presents the consecutive access vector memory (CAVM) which provides a data-parallel WLS memory architecture without duplicated data storage that imposes simple memory management computation for certain interpolation orders. We have investigated corresponding data-parallel interpolation processing for the linear and cubic cases which incorporates interpolation coefficient generation by table lookup, justified on the basis of an acceptable table memory imposition for typical phase word partitioning. We have shown how the CAVM technique is extensible to support wavetable interpolation, as distinct from phase interpolation, where indexing a set of wavetables is now performed with a fractional address in line with the SIS model.

Section (6.3) develops phase domain processing and presents a technique for generating contiguous harmonic phase sequences enabling real-time execution of the HAS model that builds on the structure introduced by Chamberlin [1976]. This architecture is germane as it allows controlled restriction of the highest harmonic according to fundamental frequency thereby eliminating the alias problem seen with WLS. We extend this architecture to synthesise non-contiguous harmonic sequences according an arbitrary integer harmonic multiplier parameter in addition to the usual amplitude and phase parameters. This enhancement prevents wastage of harmonic computation resource when harmonic amplitude is zero in the synthesised spectrum as seen with the contiguous harmonic form.

We extend the HAS model to support PWL partial frequency envelopes according a PAS model where partial frequencies are a *fractional multiple* of the fundamental

frequency. This architecture is attractive since partial frequency envelopes are applied from a "harmonic baseline" relative to the fundamental frequency using a *fractional multiplier* component to effect "fine tuning" of partial frequency. For completeness, we present a modified arithmetic model and processing architecture which specifies partial frequency envelopes according to a *frequency offset* model. Pipelined arithmetic architectures are presented whose performance is limited by parameter access time from the dual-port memory elements which separates host parameter update from the synthesis processing. However, reported FPGA logic and pipelined dual-port memory speeds indicate that a 5 ns elemental computation time is achievable and corresponds to over 4000 partials using a pipelined processor at 48 kHz sample rate.

7.3 Limitations and Areas for Further Investigation

Aliasing within the WLS model remains a fundamental problem worthy of further investigation. Limiting N_h so that $N_h\varphi > 2^{M-1}$ combined with oversampling the phase-accumulation process appears the only means of restricting this problem with the present model architecture. Applying the MWS model with manifold distinct wavetables containing progressively reduced upper harmonics across the frequency range is one possible line of attack. The reducing cost of high density memory offsets the large number of wavetables required in the MWS set. However, wavetable computation and update time must be considered.

Interpolation coefficient generation within the interpolated WLS model as presented in section (6.2.4), incurs a lookup table memory size which is linearly proportional to interpolation order and exponentially related to phase fraction field width. This area is worthy of further investigation to identify redundancies in the coefficient definition expressions which could be exploited to reduce lookup table length. Furthermore, it is evident that for a given interpolation order, some coefficient ranges are significantly

less than unity which motivates optimisation of multiplier input word length to reduce gate count in FPGA or VLSI implementations.

Further work is needed to establish the efficacy of a fractional multiplier or frequency offset PAS processing model as presented in section (6.3). The suitability of a particular technique is largely dependent on whether partial frequency deviations from a harmonic baseline are optimally specified by a *frequency multiple* or *frequency offset* envelope and suggests further detailed investigation of natural instrument spectrum behaviour at various pitches across their musical range.

The length of the pitch to phase-increment translation table is reducible by observing that for equally tempered tuning there is an *exact* ratio of two between corresponding values in *adjacent* octave data blocks within the table. This suggests that table length can be significantly reduced by storing only the lowest octave of data and adding arithmetic shifting logic to generate higher octave values which can be incorporated into the pipelined architecture presented to offset increased computation time. PWL envelope generation has not been investigated in the architectural models presented in section (6.3) and may be implemented as an additional pipeline "processing layer" between the host interface and the sample computation partitioned by DPM blocks.

Bibliography

[Ackenhusen, 1999] Ackenhusen, J. G. *Real-Time Signal Processing – Design and Implementation of Signal Processing Systems*. Prentice Hall PTR, 1999.

[Borin *et al*, 1997] Borin, G., De Ploi, G. and Sarti, A. *Musical Signal Synthesis*. Musical Signal Processing, Swets and Zeitlinger B. V., 1997.

[Chamberlin, 1985] Chamberlin, H. *Musical Applications of Microprocessors*. Hayden Books, Howard W. Sams & Co., 2nd edition, 1985.

[Crochiere and Rabiner, 1983] Crochiere, R. E. and Rabiner, L. R. *Multirate Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ.

[Eargle and Foreman, 2002] Eargle, J. and Foreman, C. *JBL Audio Engineering for Sound Reinforcement*. Music Sales Ltd, ISBN 0634043552.

[Grey, 1975] Grey, J. M. *An Exploration of Musical Timbre*. Ph.D. dissertation, Department of Psychology, Stanford University, 1975.

[Helmholtz, 1863] Helmholtz. H. L. F. v. *On the Sensations of Tone as a Physiological Basis for the Theory of Music*. Dover, New York, NY, 1954.

[Ifeachor and Jervis, 2002] Ifeachor, E. C. and Jervis, B. W. *Digital Signal Processing – A Practical Approach*. 2nd edition, Prentice-Hall Inc., 2002.

[Jensen, 1999] Jensen, K. *Timbre Models of Musical Sounds*. Ph.D. dissertation, DIKU Report 99/7, 1999.

[Lyons, 2004]: Lyons, R. G. *Understanding Digital Signal Processing*. 2nd edition, Prentice Hall, 2004.

[Massie, 1998] Massie, D. C. *Wavetable Sampling Synthesis*. Applications of Digital Signal Processing to Audio and Acoustics, Kluwer Academic Publishers, 1998.

[Mathews, 1969] Mathews, M. V. *The Technology of Computer Music*. Cambridge, MA: MIT Press, 1969.

[Moore, 1990] Moore, F. R. *Elements of Computer Music*, Prentice-Hall Inc., 1990.

[Nakamura, 1996] Nakamura, S. *Numerical Analysis and Graphic Visualisation with MATLAB*. Prentice Hall PTR, 1996.

[Oppenheim & Schafer, 1975] Oppenheim, A. V. and Schafer, R. W. *Digital Signal Processing*. Prentice-Hall, 1975.

[Orfanidis, 1996] Orfanidis, S. J. *Introduction to Signal Processing*. Prentice-Hall Inc., 1996.

[Parhami, 2000] Parhami, B. *Computer Arithmetic – Algorithms and Hardware Designs*. Oxford University Press, Inc., 2000.

[Phillips, 1997] Phillips, D. K. *Algorithms and Architectures for the Multirate Additive Synthesis of Musical Tones*. PhD thesis, School of Engineering, Durham University, UK.

[Pirsch, 1996] Pirsch, P. *Architectures for Digital Signal Processing*. John Wiley and Sons, 1996.

[Proakis and Manolakis, 1996] Proakis, J. G. and Manolakis, D. G. *Digital Signal Processing Principles, Algorithms and Applications*. 3rd Edition, Prentice-Hall, Inc., 1996.

[Quatieri and McAuley, 1998] Quatieri, T. F. and McAuley, R. J. *Audio Signal Processing Based on Sinusoidal Analysis/Resynthesis*. Applications of Digital Signal Processing to Audio and Acoustics, Kluwer Academic Publishers, 1998.

[Rabiner and Gold, 1975] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*. Prentice-Hall Inc.

[Risset and Wessel, 1982] Risset, J.-C. and Wessel, D. *Exploration of Timbre by Analysis and Synthesis*. Psychology of Music, Orlando: Academic Press, 1982.

[Risset, 1969] Risset, J.-C. *Catalog of Computer-synthesized Sound*. Murray Hill: Bell Telephone Laboratories.

[Roederer, 1973] Roederer, J. *Introduction to the Physics and Psychoacoustics of Music*. The English Universities Press, London, 1973.

[Rossing, 1990] Rossing, T. D. *The Science of Sound*. 2nd Edition, Addison-Wesley, 1990.

[Serra, 1997] Serra, M.-H. *Introducing the Phase Vocoder*. Musical Signal Processing, Swets and Zeitlinger B. V., 1997.

[Symons, 1995] Symons, P. R. *A System Architecture Supporting the Concurrent Operation and Programmable Interconnection of Multiple DSP Modules for Digital Music Synthesis*. M4046831 95/01/02 Open University T401 project dissertation, 1995.

[Vaseghi, 1996] Vaseghi, S. V. *Advanced Signal Processing and Digital Noise Reduction*. John Wiley and Sons Ltd. And B. G. Teubner, 1996.

[Winckel, 1967] Winckel, F. *Music, Sound and Sensation*. Dover Publications Inc., New York.

[Zölzer, 1997] Zölzer, U. *Digital Audio Signal Processing*. Wiley, 1997.

References

[Abu-El-Haija *et al*, 1986] Abu-El-Haija, A. I. and Al-Ibrahim, M. M. *Improving Performance of Digital Sinusoidal Oscillators by Means of Error Feedback.* IEEE Transactions on Circuits and Systems, Vol. CAS-33, No. 4, April 1986.

[Alles, 1980] Alles, H. *Musical Synthesis using Real-time Digital Techniques.* Proceedings of the IEEE, 68(4).

[Alonso *et al*, 1977] Alonso, S, Appleton, J. and Jones, C. *A Computer System for every University: the Dartmouth College Example.* Creative Computing, 3(2), 1977.

[Anderson and Jensen, 2001] Anderson, T. and Jensen, K. *Phase modelling of instrument sounds based on psycho acoustic experiments.* Workshop on current research directions in computer music, Barcelona, Spain, 2001.

[Arfib, 1979] Arfib, D. *Digital Synthesis of Complex Spectra by means of Multiplication of Non-linear Distorted Sine Waves.* Journal of the Audio Engineering Society, 27(10), 1979.

[Asanovic *et al*, 1995] Asanovic, K., Kingsbury, B., Irrisou, B., Beck, J., Wawrzynek, J. *T0: A Single-Chip Vector Microprocessor with Reconfigurable Pipelines.* Proceedings of the 22nd European Solid-State Circuits Conference, September, 1996.

[Borch & Sundberg, 2002] Borch, D. Z. and Sundberg, J. *Spectral Distribution of Solo Voice and Accompaniment in Pop Music.* Speech, Music and Hearing, KTH, Stockholm, Sweden, TMH-QPSR Vol. 43, 2002.

[Chamberlin, 1976] Chamberlin, H. A. *Experimental Fourier Series Tone Generator.* Journal of the Audio Engineering Society, Vol. 24, No. 4, May 1976.

[Chowning, 1973] Chowning, J. *The Synthesis of Complex Audio Spectra by means of Frequency Modulation.* Journal of the Audio Engineering Society, 21(7), 1973.

[Comerford, 1993] Comerford, P. J. *Simulating an Organ with Additive Synthesis.* Computer Music Journal 17(2).

[Curticapean *et al*, 2000] Curticapean, F., Palomäki, K. and Niittylahti, J. *Hardware Implementation of a Quadrature Digital Oscillator.* Proceedings of NORSIG, Kolmården, Sweden June 2000.

[Dannenberg, 1998] Dannenberg, R. B. *Interpolation Error in Waveform Table Lookup.* Proceedings of the 1998 International Computer Music Conference.

[De Poli, 1983] De Poli, G. *A Tutorial on Digital Sound Synthesis Techniques.* Computer Music Journal, 7(4), 1983.

[Freed *et al*, 1993] Freed, A., Rodet, X. and Depalle, P. *Performance, Synthesis and Control of Additive Synthesis on a Desktop Computer using FFT*$^{-1}$. Proceedings of the 19th International Computer Music Conference, Waseda University Centre for Scholarly Information, 1993.

[Furuno *et al*, 1975] Furuno, K., Mitra, S. K., Hirano, K. and Ito, Y. *Design of Digital Sinusoidal Oscillators with Absolute Periodicity.* IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-11, November 1975.

[Goodwin and Kogon, 1995] Goodwin, M. and Kogon, A. *Overlap-Add Synthesis of Nonstationary Sinusoids.* Proceedings of the 1995 International Computer Music Conference, Canada.

[Goodwin and Rodet, 1994] Goodwin, M. and Rodet, X. *Efficient Fourier Synthesis of Nonstationary Sinusoids.* Proceedings of the 1994 International Computer Music Conference, Aarhus, Denmark.

[Gordon and Smith, 1985] Gordon, J. W. and Smith, J. O. *A Sine Generation Algorithm for VLSI Applications.* Proceedings of the 1985 International Computer Music Conference, Vancouver.

[Grey and Moorer, 1977] Grey, J. M. and Moorer, J. A. *Perceptual Evaluations of Synthesized Musical Instrument Tones.* Journal of the Acoustic Society of America, 63, pp. 454-462, 1977.

[Haken, 1991] Haken, L. *Computational Methods for Real-time Fourier Synthesis.* IEEE Transactions on Signal Processing, Vol. 40, No. 2, 1991.

[Harris, 1978] Harris, F. J. *On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform.* Proceedings of the IEEE, Vol. 66, No. 1, January 1978.

[Hodes and Freed, 1999] Hodes, T. and Freed, A. *Second-order Recursive Oscillator for Musical Additive Synthesis.* Proceedings of the 1999 International Computer Music Conference, Beijing, China.

Available on-line at: http://www.cnmat.berkeley.edu/~adrian/.

[Hodes *et al*, 1999] Hodes, T., Hauser, J., Freed, A., Wawrzynek, J. and Wessel, D. *A Fixed-Point Recursive Digital Oscillator for Additive Synthesis of Audio.* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, March 15-19, 1999.

[Horner *et al*, 1993] Horner, A., Beauchamp, J., Haken, L. *Methods for Multiple Wavetable Synthesis of Musical Instrument Tones*. Journal of the Audio Engineering Society, 41(5), pp. 336-356, 1993.

[Jaffe and Smith, 1983] Jaffe, D. and Smith, J. O. *Extensions of the Karplus-Strong Plucked String Algorithm*. Computer Music Journal, 7(2).

[Jaffe, 1995] Jaffe, D. A. *Ten Criteria for Evaluating Synthesis Techniques*. Computer Music Journal, 19(1), pp. 76-87.

[Jansson & Sundberg, 1976] Jansson, E. V. and Sunberg, J. *Long Time Average Spectra Applied to Analysis of Music – Method and Application*. Acustica 34, 1976.

[Karplus and Strong, 1983] Karplus, R. and Strong, A. *Digital Synthesis of Plucked String and Drum Timbres*. Computer Music Journal, 7(2), 1983.

[Kleczowski, 1989] Kleczowski, P. *Group Additive Synthesis*. Computer Music Journal 13(1).

[Lane *et al*, 1997] Lane, J., Hoory, D., Martinez, E. and Wang, P. *Modeling Analog Synthesis with DSPs*. Computer Music Journal, 21(4), 1997.

[Loy, 1981] Loy, D. G. *Notes on the implementation of MUSBOX: A compiler for the Systems Concepts digital synthesizer*. In The Music Machine, pp. 333-349, Cambridge MA: MIT Press, 1981.

[Madisetti *et al*, 1999] Madisetti, A., Kwentus, A. Y. and Willson, A. N. *A 100-MHz, 16-b, Direct Digital Frequency Synthesiser with a 100-dBc Spurious-Free Dynamic Range*. IEEE Journal of Solid-State Circuits, Vol. 34, No. 8, August, 1999.

[Marentakis and Jensen, 2002] Marentakis, G. and Jensen, K. *Sinusoidal Synthesis Optimization*. Proceedings of the 2002 International Computer Music Conference, Gšteborg, Sweden.

[Massie, 1985] Massie, D. *The Emulator II Computer Music Environment*. Proceedings of the 1985 International Computer Music Conference.

[Mathews, 1963] Mathews, M. V. *The Digital Computer as a Musical Instrument*. Science, Vol. 142, No. 11, pp 553-557, 1963.

[Meade, 1982] Meade, M. L. *Advances in Lock-in Amplifiers*. Journal of Physics, Instrument Science and Technology, Vol. 15, 1982.

[Moog, 1965] Moog, R. A. *Voltage Controlled Electronic Music Modules*. Journal of the Audio Engineering Society, Vol. 13, No. 3, July 1965.

[Moore, 1977] Moore, F. R. *Table Lookup Noise for Sinusoidal Digital Oscillators.* Computer Music Journal, 1(2), 1977.

[Moorer, 1976] Moorer, J. A. *The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulas.* Journal of the Audio Engineering Society, Vol. 24, No. 9, November 1976.

[Moorer, 1978] Moorer, J. A. *The Use of the Phase Vocoder in Computer Music Applications.* Journal of the Audio Engineering Society, Vol. 26, Nos. 1/2, January/February 1978.

[Nicholas and Samueli, 1987] Nicholas, H. and Samueli, H. *An Analysis of the Output Spectrum of DDFS in the Presence of Phase-Accumulator Truncation.* 41st Annual Frequency Control Symposium, 1987.

[Nicholas *et al*, 1988] Nicholas, H., Samueli, H. and Kim, B. *The Optimization of DDFS Performance in the Presence of Finite Word Length Effects.* 42nd Annual Frequency Control Symposium, 1988.

[Rakowski, 1971] Rakowski, A. Pitch *Discrimination at the Threshold of Hearing.* Proceedings of the 7th International Congress on Acoustics, Vol. 3, Budapest.

[Risset and Mathews, 1969] Risset, J.-C. and Mathews, M. *Analysis of Musical Instrument Tones.* Physics Today, Vol. 22, No. 2.

[Risset, 1969] Risset, J.-C. *Catalog of Computer-synthesized Sound.* Murray Hill: Bell Telephone Laboratories.

[Risset, 1985] Risset, J.-C. *Computer Music Experiments: 1964 –.* Computer Music Journal, 9(1).

[Roads, 1996] Roads, C. *The Computer Music Tutorial.* The MIT Press, 1996.

[Rodet and Depalle, 1992] Rodet, X. and Depalle, P. *A New Additive Synthesis Method using Inverse Fourier Transform and Spectral Envelopes.* Proceedings of the 1992 International Computer Music Conference, San Francisco.

[Samson, 1980] Samson, P. *A General Purpose Synthesizer.* Journal of the Audio Engineering Society, 28(3).

[Sandell and Martens, 1992] Sandell, G. and Martens, W. *Prototyping and Interpolation of Multiple Musical Timbres using Principal Components-based Analysis.* Proceedings of the 1992 International Computer Music Conference, San Francisco.

[Sandell, 1994] Sandell, G. J. *SHARC Timbre Database.* Beta release 0.921, November 1994. Sussex University, U.K.

[Serra and Smith, 1990] Serra, X. and Smith, J. O. *Spectral Modelling Synthesis: A Sound Analysis/Synthesis System based on a Deterministic Plus Stochastic Decomposition.* Computer Music Journal, 14(4).

[Serra *et al*, 1990] Serra, M.-H., Rubine, D., Dannenberg, R. *Analysis and Synthesis of Tones by Spectral Interpolation.* Journal of the Audio Engineering Society, 38(3), 1990.

[Smith and Cook, 1992] Smith, J. O. and Cook, P. R. *The Second-Order Digital Waveguide Oscillator.* Proceedings of the 1992 International Computer Music Conference, San Jose. Available on-line at http://www-ccrma.stanford.edu/~jos/.

[Smith and Cook, 1992] Smith, J. O. and Cook, P. R. *The Second-Order Digital Waveguide Oscillator.* Proceedings of the 1992 International Computer Music Conference, San Jose. 1992. Available online at http://ccrma.stanford.edu/~jos/wgo/.

[Smith and Gossett, 1984] Smith, J. O. and Gossett, P. *A Flexible Sampling-Rate Conversion Method.* Proceedings of the 1984 International Conference on Acoustics, Speech and Signal Processing, San Diego, Vol. 2.

[Smith and Serra, 1987] Smith, J. O. and Serra, X. *PARSHL: An Analysis/Synthesis Program for Non-harmonic Sounds based on a Sinusoidal Representation.* Proceedings of the 1987 International Computer Music Conference, Illinois USA.

[Smith, 1987] Smith, J. O. *Waveguide Filter Tutorial.* Proceedings of the 1987 International Computer Music Conference, Champaign-Urbana.

[Smith, 1991] Smith, J. O. *Viewpoints on the History of Digital Synthesis.* Proceedings of the 1991 International Computer Music Conference (ICMC-91), Montreal, pp. 1-10, October 1991. Available on-line at http://www-ccrma.stanford.edu/~jos/.

[Smith, 1992] Smith, J. O. *Physical Modelling using Digital Waveguides.* Computer Music Journal, 16(4).

[Smith, 2004] Smith, J. O. *Physical Audio Signal Processing: Digital Waveguide Modeling of Musical Instruments and Audio Effects.* Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, 2004. Available on-line at http://ccrma.stanford.edu/~jos/pasp/.

[Snell, 1977] Snell, J. *Design of a Digital Oscillator that will Generate up to 256 Low-distortion Sine Waves in Real-time.* Computer Music Journal 1(2).

[Stapleton and Bass, 1988] Stapleton, J. C. and Bass, S. C. *Synthesis of Musical Tones based on the Karhunen-Loeve Transform.* IEEE Transactions on Acoustics, Speech and Signal Processing. Vol. ASSP-36, pp. 305-319, March 1988.

[Symons, 2002] Symons P. R. *DDFS phase mapping technique*. IEE Electronics Letters, 10th October 2002, Vol. 38, No. 21.

[Symons, 2004] Symons, P. R. *Phase-Continuous Frequency Change in the Direct-Form, Second-Order Recursive Oscillator*. Computer Music Journal, 28(4), 2004.

[Tierney *et al*, 1971] Tierney, J., Rader, C. M. and Gold, B. *A Digital Frequency Synthesizer*. IEEE Transactions on Audio and Electroacoustics, Vol. AU-19, No. 1, March 1971.

[Volder, 1959] Volder, J. *The CORDIC Trigonometric Computing Technique*. IRE Transactions on Electronic Computing, Vol. EC-8, September 1959.

[Walther, 1971] Walther, J. *A Unified Algorithm for Elementary Functions*. Joint Computer Conference Proceedings, Vol. 38, Spring 1971.

[Weisstein, 1999a] Weisstein, E. W. *Congruence*. MathWorld A Wolfram Web Resource. Available on line at http://mathworld.wolfram.com/Congruence.html.

[Weisstein, 1999b] Weisstein, E. W. *Divided Difference*. MathWorld A Wolfram Web Resource. Available on line at http://mathworld.wolfram.com/DividedDifference.html

[Wessel, 1979] Wessel, D. L. Timbre Space as a Musical Control Structure. Computer Music Journal, 3(2), 1979.

[Winckel, 1967] Winckel, F. *Music, Sound and Sensation*. Dover Publications Inc., New York.

[Zőlzer, 1997] Zőlzer, U. *Digital Audio Signal Processing*. Wiley, 1997.

[Smith, 2000] Smith, J. C. *Synchronous Dual-Port Static RAMs for DSP and Communication Application – Application Note AN-144*. Integrated Device Technology, March 2000. Available online at: http://www.idt.com/.

[Xilinx, 2004] *Virtex-4 Family Overview DS112 (v1.2)*. December, 2004. Available online at: http://www.xilinx.com/.

Appendix A Polynomial Interpolation

A-1 Introduction

This appendix reviews the mathematical foundations of polynomial interpolation, considering the Lagrange interpolation polynomial and Newton's divided difference representation. We begin by considering an arbitrary, well-behaved function, $y = f(x)$, which is given only at $(N+1)$ discrete ordered points (x_0, y_0), (x_1, y_1), ..., (x_{N-1}, y_{N-1}), (x_N, y_N), or in general, the set of points $(x_i, f(x_i))$, $i = 0,\ldots, N$. The problem is to find the unique N^{th} order polynomial which passes through (i.e. interpolates) the $(N+1)$ points and allows a value of y to be computed to a particular accuracy at *any* value of $x \in [x_0, x_N]$. (If x lies *outside* the interval on which the data is given, the process of computing $f(x)$ is known as *extrapolation.*)

The N^{th} order interpolating polynomial which approximates $f(x)$ on some interval may be expressed in power series form as:

$$p_N(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_N x^N \qquad \text{(A-1)}$$

where $p_N(x)$ is a polynomial of order N and the a_k terms with $k \in [0, N]$ are the corresponding polynomial coefficients. Eq. (A-1) and the set of known samples can be used to establish a system of $(N+1)$ linear equations with $(N+1)$ unknowns, expressed as:

$$\begin{aligned} f(x_0) &= a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 + \ldots + a_N x_0^N \\ f(x_1) &= a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \ldots + a_N x_1^N \\ &\vdots \qquad\qquad \vdots \qquad\qquad \ddots \\ f(x_N) &= a_0 + a_1 x_N + a_2 x_N^2 + a_3 x_N^3 + \ldots + a_N x_N^N \end{aligned} \qquad \text{(A-2)}$$

From Eqs. (A-2) the polynomial coefficients are given by:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ 1 & x_2 & x_2^2 & \cdots & x_2^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^N \end{pmatrix}^{-1} \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{pmatrix} \tag{A-3}$$

where the matrix element is known as the *Vandermonde matrix*. For large N the Vandermonde matrix becomes ill-conditioned and therefore sensitive to small computational rounding errors (e.g. quantisation errors) and can easily produce inaccurate results. The Lagrange method provides higher computational efficiency compared to the power series form and is generally less susceptible to small computational errors.

A-2 The Lagrange Interpolating Polynomial

The Lagrange polynomial, $P_N(x)$, represents the polynomial approximation to $f(x)$ on the interval $x \in [x_0, x_N]$ and can be expressed as a linear combination of N^{th} order basis polynomials, $p_k(x)$, thus [Vaseghi, 1996]:

$$P_N(x) = \sum_{k=0}^{N} p_k(x) f(x_k) \tag{A-4}$$

where $p_k(x) = \dfrac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_N)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_N)} = \prod_{\substack{j=0 \\ j\neq k}}^{N} \dfrac{(x-x_j)}{(x_k-x_j)}$.

Hence, from Eq. (A-4) we obtain:

$$P_N(x) = \sum_{k=0}^{N} \left[f(x_k) \prod_{\substack{j=0 \\ j\neq k}}^{N} \frac{(x-x_j)}{(x_k-x_j)} \right] \tag{A-5}$$

The k^{th} Lagrange polynomial coefficient, $p_k(x)$, is unity at the k^{th} given data point ($p_k(x_k) = 1$) and zero at every other given data point ($p_k(x_j) = 0, \quad k \neq j$). Hence, $P_N(x_k) = p_k(x_k) f(x_k) = f(x_k) = y_k$ and so the polynomial passes through the given

data points as required. For example, with $N=3$ (as pertinent to the discussion of section (5.2)) we have:

$$f(x)=\frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}y_0+\frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}y_1 \\ +\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}y_2+\frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}y_3 \tag{A-6}$$

An important question arises at this point – *for a given order polynomial, how does the error vary with the independent variable, x?* The error is defined by [Nakamura, 1996]:

$$\varepsilon(x)=f(x)-P_N(x) \tag{A-7}$$

where $f(x)$ represents the exact function being interpolated by the polynomial, $P_N(x)$, on the interval $x\in[x_0, x_N]$. A generalised analytic expression of the error is given by [Nakamura, 1996]:

$$\varepsilon(x)=L_N(x)f^{(N+1)}(\xi) \tag{A-8}$$

where $f^{(N+1)}(x)$ denotes the $(N+1)^{\text{th}}$ derivative of $f(x)$, ξ depends on x with $\xi\in[x_0, x_N]$ and:

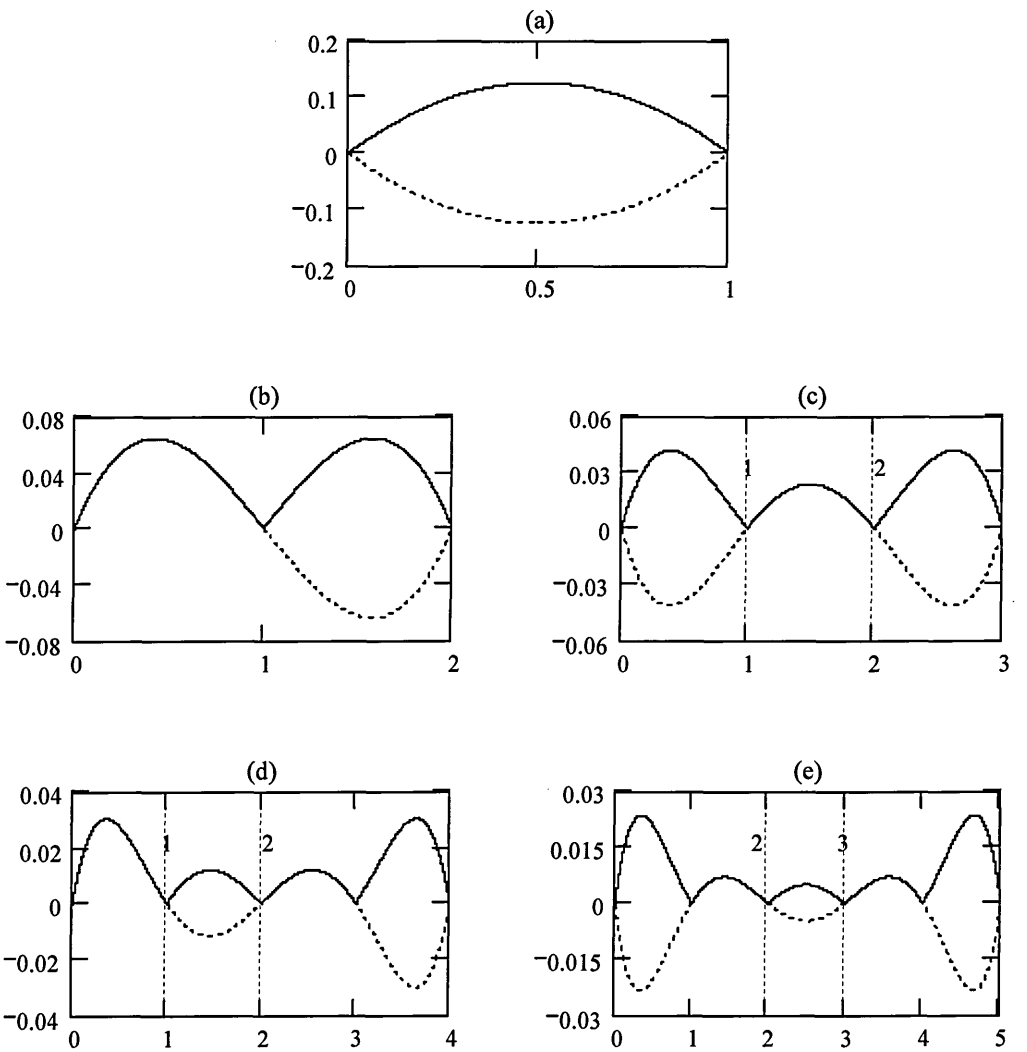
$$L_N(x)=\frac{(x-x_0)(x-x_1)\cdots(x-x_{N-1})(x-x_N)}{(N+1)!} \tag{A-9}$$

We define an upper bound on the magnitude of $\varepsilon(x)$, thus: [Nakamura, 1996]

$$|\varepsilon(x)|\le|L_N(x)|\max_{x_0\le\xi\le x_N}|f^{(N+1)}(\xi)| \tag{A-10}$$

The second term in the right hand side of Eq. (A-10) is constant on the whole domain and so the behaviour of the upper bound on $|\varepsilon(x)|$ is determined by $L_N(x)$. Figures (A-1a) thru (A-1e) illustrate the behaviour of the $|L_N(x)|$ and $L_N(x)$ polynomials for $N\in[1, 5]$ and unit-spaced tabulation points.

We observe that $|L_N(x)|$ exhibits minimum oscillation amplitude on the *middle* sub-interval of the range $[x_0, x_N]$ for odd N (i.e. $\left[\left\lfloor \frac{x_N}{2} \right\rfloor, \left\lceil \frac{x_N}{2} \right\rceil\right]$ for unit x increments) and on the two middle intervals centred about the middle sample for even N. In general, this condition is true for all Lagrangian interpolation polynomials [Nakamura, 1996].



Figures (A-1a) through (A-1e): Behaviour of the $L_N(x)$ (dashed line) and $|L_N(x)|$ (solid line) polynomials for $N \in [1, 5]$ corresponding with Figures (a) thru (e), respectively.

For odd N, the fractional address should therefore be placed in the middle interval of the interpolation data set for minimum error bound. For even N, either of the two central sub-intervals gives minimum error magnitude bound.

Returning to our example cubic interpolation polynomial and assuming unit-spaced x values, Eq. (A-5) places x in the first sub-interval, $[x_0, x_1]$, of the data set $\{f(x_0), f(x_1), f(x_2), f(x_3)\}$, leading to *non-optimal* error bound. Introducing an offset to the x subscripts in Eq. (A-5) so that the i^{th} subscript becomes $x_{\left(i-\left\lfloor \frac{N-1}{2} \right\rfloor\right)}$, places x in the middle sub-interval, $[x_0, x_1]$, of the data set $\{f(x_{-1}), f(x_0), f(x_1), f(x_2)\}$ as illustrated in Figure (A-2) and realises *minimum* interpolation error bound.

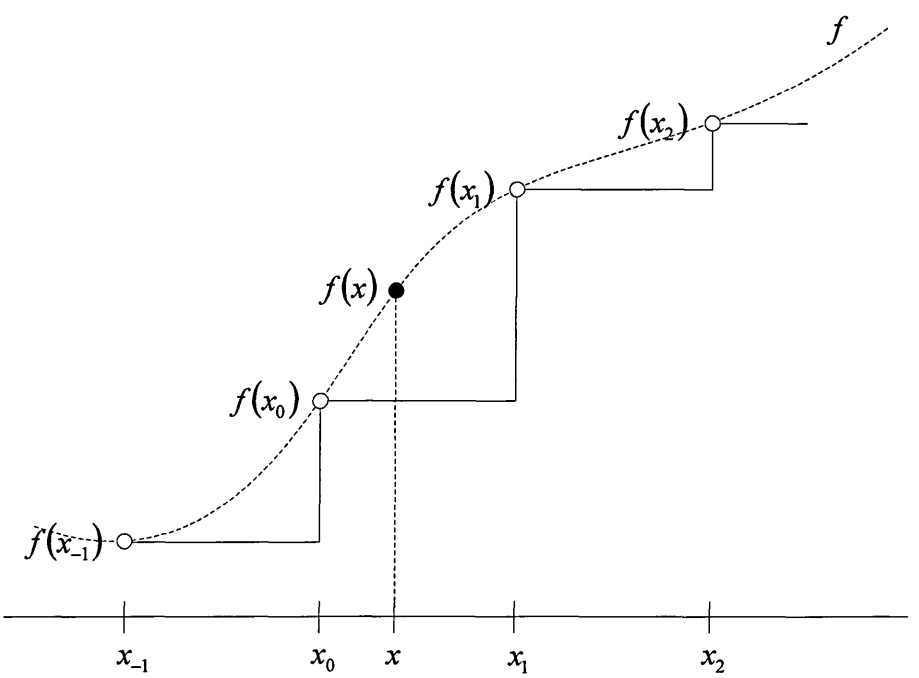


Figure (A-2): Optimum data set and sub-interval for the ***cubic*** *interpolating polynomial assuming unit-spaced x values.*

A-3 The Newton Interpolating Polynomial

Lagrange interpolation polynomials can be reformulated in *Newton form* which have a recursive structure allowing an order N polynomial to be constructed by extension of an order $(N-1)$ polynomial, which we illustrate with the examples:

zero-order polynomial: $P_0(x) = a_0 = f(x_0)$

linear polynomial:
$$\begin{aligned} P_1(x) &= a_0 + a_1(x - x_0) \\ &= P_0(x) + a_1(x - x_0) \end{aligned}$$

quadratic polynomial:
$$\begin{aligned} P_2(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) \\ &= P_1(x) + a_2(x - x_0)(x - x_1) \end{aligned}$$

cubic polynomial:
$$\begin{aligned} P_3(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)\ldots \\ &\quad + a_3(x - x_0)(x - x_1)(x - x_2) \\ &= P_2(x) + a_3(x - x_0)(x - x_1)(x - x_2) \end{aligned}$$

In general, we express the Newton interpolation polynomial with the recursion [Vaseghi, 1996]:

$$\begin{aligned} P_N(x) &= P_{N-1}(x) + a_m(x - x_0)(x - x_1)\cdots(x - x_{N-1}) \\ &= P_{N-1}(x) + a_m \prod_{j=0}^{N-1}(x - x_j) \end{aligned} \tag{A-11}$$

where a_m represents the m^{th} *divided difference* of the tabulated function points given by [Weisstein, 1999b]:

$$a_m = \sum_{k=0}^{m}\left(f(x_k) \prod_{\substack{j=0 \\ j \neq k}}^{m} (x_k - x_j)^{-1} \right) \tag{A-12}$$

$$m \in [0, N]$$

A-4 Horner's Algorithm

Horner's algorithm [Orfanidis, 1996] expresses the generalised N^{th} order polynomial given by:

$$P(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{N-1} x^{N-1} + c_N x^N \qquad \text{(A-13)}$$

in a reduced-multiplier form that requires only N multiplication and addition operations, thus:

$$P(x) = c_0 + (c_1 + (c_2 + \ldots (c_{N-1} + c_N x)x)x)x \qquad \text{(A-14)}$$

where the c terms denote the $(N+1)$ polynomial coefficients. Horner's algorithm may be generalised by an iterative program whose pseudo-code is given by:

$$\begin{aligned} &\textbf{Initialise} \quad v \leftarrow c_N \\ &\textbf{For} \quad k = (N-1), \cdots, 0 \\ &\quad v \leftarrow c_k + vx \\ &\textbf{Return} \quad v \end{aligned} \qquad \text{(A-15)}$$

Horner's algorithm is of interest here since the Horner-factorised Newton interpolation polynomial imposes a reduced multiplication overhead compared to the Lagrange polynomial. However, the $(N+1)$ divided difference terms require linear combination of $(N+1)$ samples in the interpolation set with constant weighting terms.

Appendix B Performance Simulation

B-1 Introduction

This appendix presents numerical models that have been developed within this research to simulate performance metrics specific to the WLS, TIPM, HAS and PAS processing models. The models are written as Mathcad 11.2a function scripts[1] using the Mathcad programming syntax and use three particular sub-functions:

- The *rounding* quantisation function $\mathrm{QR}(x,q) := q \cdot \mathrm{round}\left(\frac{x}{q}\right)$ quantises a variable x with quantisation interval q and simulates fixed-point arithmetic behaviour in the otherwise full-precision floating-point Mathcad environment.
- The unit-amplitude, piecewise-linear (PWL) amplitude envelope function $A(h, r_1, r_2, b) := \mathrm{if}\left[h < b, \left(\frac{1}{h}\right)^{r_1}, \left(\frac{1}{b}\right)^{(r_1 - r_2)} \cdot \left(\frac{1}{h}\right)^{r_2}\right]$ returns the amplitude of a partial, h, assuming a two segment PWL response given by Eq. (5.4.11). Segment slopes are controlled by parameters r_1 and r_2, with slope given by $-6r_1$ or $-6r_2$ dB/octave and PWL slope breakpoint at partial b.
- The function $\mathrm{modn}(x, y) := \mathrm{if}(x < 0, y + x, \mathrm{mod}(x, y))$ returns the modulo y value of a *negative* x argument as $(y + x)$, with positive arguments handled as normal. Interpolated WLS often requires indexing a *negative* wavetable location which we interpret as a modulo "wrap-around" assuming the wavetable end-points are exactly phase-continuous.

[1] Mathcad program listings use non-italicised variables. However, in this appendix we denote Mathcad variables using italicised mathematical notation to avoid confusion with general discussion text.

B-2 The SNR_interp_R Function

The Mathcad function `SNR_interp_R` listed in Figure B-1 computes SNR according to Eq. (5.4.3) for the interpolated WLS processing model and is called with the function expression: SNR_interp_R($N, PR, M, I, \psi, R, bits, N_s, N_h, r_1, r_2, b$).

Parameter denotations follow their normal definition as given in the Glossary, with the *phase rounding* parameter PR selecting full phase truncation when $PR = 0$ and rounded phase truncation when $PR = 1$. Phase increment (and hence frequency) is denoted by ψ with *bits* denoting the amplitude sample word size in bits. Parameters r_1, r_2 and b specify the spectrum envelope response used to determine the wavetable harmonic amplitude values. This function may be partitioned into distinct stages which are functionally similar across all of the scripts presented in this Appendix.

We first initialise quantisation variables, q_1 and q_2, which represent single and double precision quantisation intervals, respectively. We also define ε as the smallest number that can be represented within the Mathcad floating-point environment.

$$\begin{aligned} &\text{result} \leftarrow 0 \\ &q1 \leftarrow \frac{1}{2^{\text{bits}-1}} \\ &q2 \leftarrow \frac{1}{2^{(2\cdot \text{bits})-1}} \\ &\varepsilon \leftarrow \frac{1}{\infty} \end{aligned}$$

Offsetting potentially zero denominator and log arguments by ε prevents divide-by-zero and log(0) errors. We next compute the harmonic amplitude vector, a, according to a PWL spectrum response and a reference waveform vector, `wav_ref`, using N_h harmonics over N_s samples.

$$
\begin{aligned}
&\text{for } h \in 1..\,Nh \\
&\quad a_h \leftarrow A(h, r1, r2, b) \\
&\text{for } n \in 0..\,Ns-1 \\
&\quad wav\_ref_n \leftarrow \sum_{k=1}^{Nh}\left[a_k \cdot \sin\left[2\cdot k\cdot \pi \cdot \frac{\mathrm{mod}\left[(n\cdot\psi), 2^M\right]}{2^M}\right]\right] \\
&wav\_ref \leftarrow \frac{wav\_ref}{\max(wav\_ref)}
\end{aligned}
$$

The vector `wav_ref` is computed to full floating-point precision and normalised to unit-amplitude. We then compute the wavetable vector `wav_lut` of length 2^I samples, normalise to unit-amplitude and quantise to q_1 fixed-point precision.

$$
\begin{aligned}
&\text{for } n \in 0..\,2^I-1 \\
&\quad wav\_lut_n \leftarrow \sum_{k=1}^{Nh}\left(a_k \cdot \sin\left(2\cdot k\cdot \pi \cdot \frac{n}{2^I}\right)\right) \\
&wav\_lut \leftarrow \frac{wav\_lut}{\max(wav\_lut)} \\
&\text{for } n \in 0..\,2^I-1 \\
&\quad wav\_lut_n \leftarrow QR\left(wav\_lut_n, q1\right)
\end{aligned}
$$

In the program kernel we compute the order N interpolated wavetable lookup vector `wav_interp` of length N_s using results from Chapter 5^2. If $PR = 1$ *rounded* phase truncation is applied and the value of N ignored, otherwise full phase truncation is applied when $PR = 0$.

$$
\begin{aligned}
&\text{for } n \in 0..\,Ns-1 \\
&\quad \text{if}\left[PR = 1, \phi t_n \leftarrow \mathrm{mod}\left[\mathrm{round}\left[\frac{\mathrm{mod}\left[(n\cdot\psi), 2^M\right]}{2^{M-I}}\right], 2^I\right], \phi t_n \leftarrow \mathrm{floor}\left[\frac{\mathrm{mod}\left[(n\cdot\psi), 2^M\right]}{2^{M-I}}\right]\right] \\
&\quad \text{if}\left[PR = 1, \alpha_n \leftarrow 1, \alpha_n \leftarrow \frac{2^R \cdot \mathrm{mod}\left[\mathrm{floor}\left[\frac{\mathrm{mod}\left[(n\cdot\psi), 2^M\right] - \left(2^{M-I}\right)\cdot \phi t_n}{2^R}\right], 2^{M-I-R}\right]}{2^{M-I}}\right] \\
&\quad \text{if}\left[PR = 1, wav\_interp_n \leftarrow wav\_lut_{(\phi t_n)}, wav\_interp_n \leftarrow \sum_{k=0}^{N}\left[wav\_lut_{\mathrm{mod}_n\left[\left(\phi t_n - \mathrm{floor}\left(\frac{N-1}{2}\right)+k\right), 2^I\right]} \cdot QR\left[\prod_{j=0}^{N}\left[\text{if}\left[j = k, 1, \left(\frac{\alpha_n + \mathrm{floor}\left(\frac{N-1}{2}\right) - j}{k-j}\right)\right], q2\right]\right]\right] \\
&\quad wav\_interp_n \leftarrow QR\left(wav\_interp_n, q1\right)
\end{aligned}
$$

Interpolation computations are performed using Eq. (5.4.5) with multiply-accumulate operations computed to q_2 fixed-point precision, quantising back to q_1 precision upon completion. Finally, we normalise `wav_interp` and compute the error vector `interp_error` relative to the full precision vector `wav_ref`.

$$\begin{aligned}
&\text{wav_interp} \leftarrow \frac{\text{wav_interp}}{\max(\text{wav_interp})} \\
&\text{interp_error} \leftarrow (\text{wav_interp} - \text{wav_ref}) \\
&\text{SNR} \leftarrow 20 \cdot \log\left(\frac{\text{stdev}(\text{wav_ref})}{\text{stdev}(\text{interp_error}) + \varepsilon}\right) \\
&\text{result} \leftarrow \text{round}(\text{SNR})
\end{aligned}$$

The returned SNR result is computed by taking the rounded log of the ratio between the `wav_ref` and `interp_error` RMS values. Returned SNR values are therefore rounded to the nearest dB.

B-3 The errspec_interp_R Function

The Mathcad function `errspec_interp_R` listed in Figure (B-2) computes the error spectrum for the interpolated WLS processing model and is called with the function expression: errspec_interp_R($N, PR, M, I, \psi, R, bits, N_s, N_h, r_1, r_2, b$). This program is identical to the `SNR_interp_R` function program, except for the variable initiation and final spectrum computation stages. Program initiation includes computation of a Hamming window vector of length N_s samples and definition of the γ variable which denotes the gain of the Hamming window function and is used to normalise the output spectrum. The program kernel is identical to that of Figure (B-1) except the final stage now computes the error spectrum from the `interp_error` variable. The error vector is scaled with the window vector and then transformed by complex FFT yielding the

[2] The Mathcad variable denoted ϕt_n corresponds with $\phi_I(n)$, the integer component of the phase accumulator output sequence.

error_spec frequency domain vector. This vector is returned after complex to real conversion, window scaling correction and logarithmic conversion.

B-4 The SNR_trig_R Function

The Mathcad function SNR_trig_R listed in Figure (B-3) computes SNR according to Eq. (5.4.3) for the trigonometric identity phase mapping (TIPM) processing model presented in section (5.3) and is called with the function expression: SNR_trig_R$(M, I, \psi, bits, N_s, R)$ which has identical argument definition to the SNR_interp_R function discussed in section (B-2).

Following variable initialisation and computation of a full precision reference sinusoid vector, the four lookup tables are computed using Eqs. (5.3.6) and (5.3.7). In the program kernel, integer and fraction phase sequences are computed and used to index the four lookup tables as illustrated in Figure (5.3.2), with the phase fraction truncated by R bits[3]. Finally, SNR is computed, rounded and returned in an analogous manner to the SNR_interp_R function.

B-5 The errspec_trig_R Function

The Mathcad function errspec_trig_R listed in Figure (B-4) computes the error spectrum for the trigonometric identity phase mapping (TIPM) processing model and is called with the function expression: errspec_trig_R$(M, I, \psi, bits, N_s, R)$. This program is identical to the SNR_interp_R function program, except for the variable initiation and final spectrum computation stages.

[3] The Mathcad variables denoted ϕt_n and ϕf_n correspond with $\phi_I(n)$ and $\phi_F(n)$, respectively.

B-6 The SNR_interp_HAS Function

The Mathcad function `SNR_interp_HAS` listed in Figure (B-5) computes SNR according to Eq. (5.4.3) for the harmonic additive synthesis (HAS) processing model presented in section (2.3) and is called with the function expression: $\text{SNR\_interp\_HAS}(N, M, I, \psi, bits, N_s, N_h, r_1, r_2, b)$. This function has identical argument definition to the `SNR_interp_R` function discussed in section (B-2) and requires three specific support functions which compute harmonic multiples of the phase accumulator output value partitioning the result into integer and fraction components:

- The function: $\mu\_I(\phi, M, F, k) := \text{floor}\left(\frac{\text{mod}(k \cdot \phi, 2^M)}{2^F}\right)$ returns the I-bit integer field of the k^{th} multiple of the phase value ϕ, where F denotes the phase fraction field width in bits.
- The function: $\mu\_F(\phi, M, F, k) := \left(\text{mod}(k \cdot \phi, 2^M) - \text{floor}\left(\frac{\text{mod}(k \cdot \phi, 2^M)}{2^F}\right) \cdot 2^F\right)$ returns the F-bit fraction field of the k^{th} multiple of the phase value, ϕ.
- The function: $\mu\_\alpha(\phi, M, F, k) := \frac{\left(\text{mod}(k \cdot \phi, 2^M) - \text{floor}\left(\frac{\text{mod}(k \cdot \phi, 2^M)}{2^F}\right) \cdot 2^F\right)}{2^F}$ returns the normalised fraction value.

For $N \geq 0$, this function effects an order N polynomial phase mapping interpolation and for $N < 0$ this function effects TIPM. Otherwise, this function follows a similar structure and argument interpretation to the WLS `SNR_interp_R` function.

B-7 The errspec_interp_HAS Function

The Mathcad function `errspec_interp_HAS` listed in Figure (B-6) computes the error spectrum as discussed in section (5.4.3) for the HAS processing model and is called with the expression: errspec_interp_HAS($N, M, I, \psi, bits, N_s, N_h, r_1, r_2, b$).

The function is essentially identical to the `SNR_interp_HAS` function program except for the variable initiation and error spectrum computation stages which follow the same format as the `errspec_interp_R` function.

B-8 The SNR_interp_PAS Function

The Mathcad function `SNR_interp_PAS` listed in Figure (B-7) computes SNR according to Eq. (5.4.3) for the PAS processing model presented in section (2.3) and is called with the expression: SNR_interp_PAS($f_s, B, N, M, I, \rho, bits, N_s, N_p, r_1, r_2, b, v$).

This function has five new parameters unique to the PAS model[4]:

- f_s specifies the sample rate in Hz;
- B specifies the equally tempered tuning base frequency in Hz;
- ρ specifies fundamental *pitch* in $\frac{1}{128}$ of a semitone;
- N_p specifies the number of partials summated;
- v specifies the maximum pseudo-random value of $\beta_k(n)$.

The remaining function arguments are identical to the `SNR_interp_R` case presented in section (B-2). The function comprises two kernel Mathcad sub-programs which simulate the phase domain processing model of Figure (6.3.11). The first sub-program begins by computing the pitch-to-phase increment translation table according to Eq. (6.3.2) and the partial piecewise-linear amplitude vector, a. Tuning resolution is set at

[4] In this and subsequent PAS models we use γ to denote the minimum equally tempered tuning frequency ratio as distinct from the window gain factor in earlier models.

$\frac{1}{128}$ of a semitone. Next, we compute two vectors, k and β, which set the individual partial frequencies according the processing model presented in section (6.3.6). The program sets the k_h terms as a contiguous harmonic sequence (although they may be arbitrary integers provided $k_h < 2^{I-1}$ to prevent aliasing) and the β_h partial fractional multiplier terms as *pseudo-random* values with maximum value v. The fundamental β term is set to zero and hence the fundamental pitch is exact. Finally, this sub-program computes the normalised reference waveform vector, `wav_ref`.

$$\begin{aligned}
&\text{for } n \in 0..2^{14}-1 \\
&\quad \Psi_n \leftarrow \left(\frac{B\cdot\gamma^{n}\cdot 2^{M}}{fs}\right) \\
&\text{for } h \in 1..Np \\
&\quad a_h \leftarrow A(h,r1,r2,b) \\
&\text{for } h \in 1..Np \\
&\quad k_h \leftarrow h \\
&\quad \text{if}\left(h \geq 2, \beta_h \leftarrow \text{round}(\text{rnd}(v)), \beta_h \leftarrow 0\right) \\
&\text{for } n \in 0..Ns-1 \\
&\quad wav_ref_n \leftarrow \sum_{j=1}^{Np}\left[a_j\cdot\sin\left[2\cdot\pi\cdot\frac{\text{mod}\left[n\cdot\left[\frac{B\cdot\gamma^{\rho}\cdot 2^{M}}{fs}\cdot\left(\gamma^{\beta_j}+k_j-1\right)\right],2^{M}\right]}{2^{M}}\right]\right] \\
&wav_ref \leftarrow \frac{wav_ref}{\max(wav_ref)}
\end{aligned}$$

Since the difference between `wav_ref` and the simulated waveform sequence, `wav_interp`, determines the amplitude error signal, it is critical that phase coherence is preserved between these two vectors *over the length of the simulation.* To ensure phase coherence, we use full-precision pitch to phase increment translation table values and do not round to the nearest integer value as given by Eq. (6.3.2). Rounding the tabulated values causes finite frequency errors and hence a steady increase in the error

sequence magnitude over the simulation due to reference and simulated signals beating in frequency. (See section (6.3.6)).

The next sub-program computes five phase sequences for each partial at each sample point in accordance with the processing model depicted in Figure (6.3.11). The final phase sequence is partitioned into integer and fraction fields, before phase mapping using an order N interpolation model as before. The SNR is computed and the rounded value returned.

$$
\begin{aligned}
&\text{for } n \in 0..Ns-1 \\
&\quad \phi 1_n \leftarrow \operatorname{mod}\left[\left(n\cdot\Psi_\rho\right),2^M\right] \\
&\quad \text{for } h \in 1..Np \\
&\quad\quad \phi 2_h \leftarrow \operatorname{mod}\left[\left[n\cdot\Psi_{(\rho+\beta_h)}\right],2^M\right] \\
&\quad\quad \phi 3_h \leftarrow \operatorname{mod}\left(\phi 2_h-\phi 1_n,2^M\right) \\
&\quad\quad \phi 4_h \leftarrow \operatorname{mod}\left(k_h\cdot\phi 1_n,2^M\right) \\
&\quad\quad \phi 5_h \leftarrow \operatorname{mod}\left(\phi 3_h+\phi 4_h,2^M\right) \\
&\quad\quad \phi t_h \leftarrow \operatorname{floor}\left(\frac{\phi 5_h}{2^F}\right) \\
&\quad\quad \alpha_h \leftarrow \left(\frac{\phi 5_h-2^F\cdot\operatorname{floor}\left(\frac{\phi 5_h}{2^F}\right)}{2^F}\right) \\
&\quad \text{wav\_interp}_n \leftarrow \sum_{h=1}^{Np}\left[a_h\cdot\sum_{m=0}^{N}\left[\text{sin\_lut}_{\operatorname{modn}\left[\left(\phi t_h-\operatorname{floor}\left(\frac{N-1}{2}\right)+m\right),2^I\right]}\cdot QR\left[\prod_{j=0}^{N}\left[\text{if}\left[j=m,1,\left(\frac{\alpha_h+\operatorname{floor}\left(\frac{N-1}{2}\right)-j}{m-j}\right)\right],q2\right]\right]\right]\right] \\
&\quad \text{wav\_interp}_n \leftarrow QR\left(\text{wav\_interp}_n,q1\right)
\end{aligned}
$$

B-9 The spec_interp_PAS_pitch Function

The Mathcad function `spec_interp_PAS_pitch` listed in Figure (B-8) computes the amplitude spectrum for the PAS processing model with partial *fractional multiplier* control as described in section (6.3.6). The function is called with the expression:

spec_interp_PAS_pitch$(f_s, B, N, M, I, \rho, bits, N_s, N_p, r_1, r_2, b, \beta)$ where the vector β of length N_p determines the respective partial fractional multipliers $\left(\gamma^{\beta}-1\right)$.

B-10 The spec_interp_PAS_freq Function

The Mathcad function `spec_interp_PAS_freq` listed in Figure (B-9) computes the amplitude spectrum for the PAS processing model with partial *frequency offset* control as described in section (6.3.7). The function is called with the expression:

spec_interp_PAS_freq($f_s, B, N, M, I, \rho, bits, N_s, N_p, r_1, r_2, b, \beta$) where the vector β of length N_P contains the respective partial frequency offsets in Hz.

$$\text{SNR\_interp\_R}(N, PR, M, I, \psi, R, bits, Ns, Nh, r1, r2, b)$$

$$\begin{aligned}
&\text{result} \leftarrow 0 \\
&q1 \leftarrow \frac{1}{2^{bits-1}} \\
&q2 \leftarrow \frac{1}{2^{(2\cdot bits)-1}} \\
&\varepsilon \leftarrow \frac{1}{\infty} \\
&\text{for } h \in 1..Nh \\
&\quad a_h \leftarrow A(h, r1, r2, b) \\
&\text{for } n \in 0..Ns-1 \\
&\quad wav_ref_n \leftarrow \sum_{k=1}^{Nh}\left[a_k \cdot \sin\left[2\cdot k\cdot \pi \cdot \frac{\text{mod}\left[(n\cdot\psi), 2^M\right]}{2^M}\right]\right] \\
&wav_ref \leftarrow \frac{wav_ref}{\max(wav_ref)} \\
&\text{for } n \in 0..2^I-1 \\
&\quad wav_lut_n \leftarrow \sum_{k=1}^{Nh}\left(a_k \cdot \sin\left(2\cdot k\cdot \pi \cdot \frac{n}{2^I}\right)\right) \\
&wav_lut \leftarrow \frac{wav_lut}{\max(wav_lut)} \\
&\text{for } n \in 0..2^I-1 \\
&\quad wav_lut_n \leftarrow QR(wav_lut_n, q1) \\
&\text{for } n \in 0..Ns-1 \\
&\quad \left[\text{if}\left[PR = 1, \phi t_n \leftarrow \text{mod}\left[\text{round}\left[\frac{\text{mod}\left[(n\cdot\psi), 2^M\right]}{2^{M-I}}\right], 2^I\right], \phi t_n \leftarrow \text{floor}\left[\frac{\text{mod}\left[(n\cdot\psi), 2^M\right]}{2^{M-I}}\right]\right]\right. \\
&\quad \left[\text{if}\left[PR = 1, \alpha_n \leftarrow 1, \alpha_n \leftarrow \frac{2^R \cdot \text{mod}\left[\text{floor}\left[\frac{\text{mod}\left[(n\cdot\psi), 2^M\right] - \left(2^{M-I}\right)\cdot \phi t_n}{2^R}\right], 2^{M-I-R}\right]}{2^{M-I}}\right]\right. \\
&\quad \left[\text{if}\left[PR = 1, wav_interp_n \leftarrow wav_lut_{(\phi t_n)}, wav_interp_n \leftarrow \sum_{k=0}^{N}\left[wav_lut_{\text{modn}\left[\left(\phi t_n - \text{floor}\left(\frac{N-1}{2}\right)+k\right), 2^I\right]} \cdot QR\left[\prod_{j=0}^{N}\text{if}\left[j = k, 1, \left(\frac{\alpha_n + \text{floor}\left(\frac{N-1}{2}\right) - j}{k-j}\right)\right], q2\right]\right]\right]\right. \\
&\quad wav_interp_n \leftarrow QR(wav_interp_n, q1) \\
&wav_interp \leftarrow \frac{wav_interp}{\max(wav_interp)} \\
&interp_error \leftarrow (wav_interp - wav_ref) \\
&SNR \leftarrow 20\cdot\log\left(\frac{stdev(wav_ref)}{stdev(interp_error) + \varepsilon}\right) \\
&\text{result} \leftarrow \text{round}(SNR)
\end{aligned}$$

Figure (B-1): The Mathcad function `SNR_interp_R`.

errspec_interp_R (N,PR,M,I,ψ,R,bits , Ns, Nh,r1,r2,b)

result ← 0

win ← hamming(Ns)

γ ← 0.54

$q1 \leftarrow \frac{1}{2^{bits-1}}$

$q2 \leftarrow \frac{1}{2^{(2\cdot bits)-1}}$

$\varepsilon \leftarrow \frac{1}{\infty}$

for $h \in 1..Nh$

$a_h \leftarrow A(h,r1,r2,b)$

for $n \in 0..Ns-1$

$wav\_ref_n \leftarrow \sum_{k=1}^{Nh}\left[a_k\cdot\sin\left[2\cdot k\cdot\pi\cdot\frac{mod\left[(n\cdot\psi),2^M\right]}{2^M}\right]\right]$

$wav\_ref \leftarrow \frac{wav\_ref}{max(wav\_ref)}$

for $n \in 0..2^I-1$

$wav\_lut_n \leftarrow \sum_{k=1}^{Nh}\left(a_k\cdot\sin\left(2\cdot k\cdot\pi\cdot\frac{n}{2^I}\right)\right)$

$wav\_lut \leftarrow \frac{wav\_lut}{max(wav\_lut)}$

for $n \in 0..2^I-1$

$wav\_lut_n \leftarrow QR\left(wav\_lut_n,q1\right)$

for $n \in 0..Ns-1$

if $\left[PR=1,\phi t_n \leftarrow mod\left[round\left[\frac{mod\left[(n\cdot\psi),2^M\right]}{2^{M-I}}\right],2^I\right],\phi t_n \leftarrow floor\left[\frac{mod\left[(n\cdot\psi),2^M\right]}{2^{M-I}}\right]\right]$

if $\left[PR=1,\alpha_n \leftarrow 1,\alpha_n \leftarrow \frac{2^R\cdot mod\left[floor\left[\frac{mod\left[(n\cdot\psi),2^M\right]-\left(2^{M-I}\right)\cdot\phi t_n}{2^R}\right],2^{M-I-R}\right]}{2^{M-I}}\right]$

if $\left[PR=1,wav\_interp_n \leftarrow wav\_lut\left(\phi t_n\right),wav\_interp_n \leftarrow \sum_{k=0}^{N}\left[wav\_lut_{modn\left[\left(\phi t_n-floor\left(\frac{N-1}{2}\right)+k\right),2^I\right]}\cdot QR\left[\prod_{j=0}^{N}if\left[j=k,1,\left(\frac{\alpha_n+floor\left(\frac{N-1}{2}\right)-j}{k-j}\right)\right],q2\right]\right]\right]$

$wav\_interp_n \leftarrow QR\left(wav\_interp_n,q1\right)$

$wav\_interp \leftarrow \frac{wav\_interp}{max(wav\_interp)}$

$interp\_error \leftarrow (wav\_interp - wav\_ref)$

$error\_spec \leftarrow mag\left(CFFT\left(\overrightarrow{(win\cdot interp\_error)}\right)+\varepsilon\right)$

for $n \in 0..Ns-1$

$error\_spec_n \leftarrow 20\cdot\log\left(2\cdot\frac{1}{\gamma}\cdot error\_spec_n\right)$

result ← error_spec

Figure (B-2): The Mathcad function `errspec_interp_R`.

$$
\begin{aligned}
&\text{SNR\_trig\_R}(M, I, \psi, \text{bits}, Ns, R) := \\
&\quad \text{result} \leftarrow 0 \\
&\quad q1 \leftarrow \frac{1}{2^{\text{bits}-1}} \\
&\quad q2 \leftarrow \frac{1}{2^{(2\cdot\text{bits})-1}} \\
&\quad \varepsilon \leftarrow \frac{1}{\infty} \\
&\quad \text{for } n \in 0..Ns-1 \\
&\qquad \text{sin\_ref}_n \leftarrow \sin\left[2\cdot\pi\cdot\frac{\text{mod}\left[(n\cdot\psi), 2^M\right]}{2^M}\right] \\
&\quad \text{for } n \in 0..2^I-1 \\
&\qquad \text{Isin\_lut}_n \leftarrow \text{QR}\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^I}\right), q1\right) \\
&\qquad \text{Icos\_lut}_n \leftarrow \text{QR}\left(\cos\left(2\cdot\pi\cdot\frac{n}{2^I}\right), q1\right) \\
&\quad \text{for } n \in 0..2^{M-I}-1 \\
&\qquad \text{Fsin\_lut}_n \leftarrow \text{QR}\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^M}\right), q1\right) \\
&\qquad \text{Fcos\_lut}_n \leftarrow \text{QR}\left(\cos\left(2\cdot\pi\cdot\frac{n}{2^M}\right), q1\right) \\
&\quad \text{for } n \in 0..Ns-1 \\
&\qquad \phi t_n \leftarrow \text{floor}\left[\frac{\text{mod}\left[(n\cdot\psi), 2^M\right]}{2^{M-I}}\right] \\
&\qquad \phi f_n \leftarrow \text{mod}\left[(n\cdot\psi), 2^M\right] - \left(2^{M-I}\right)\cdot\phi t_n \\
&\qquad \phi f_n \leftarrow 2^R\cdot\text{mod}\left(\text{floor}\left(\frac{\phi f_n}{2^R}\right), 2^{M-I-R}\right) \\
&\qquad \text{sin\_trig}_n \leftarrow \left[\text{QR}\left[\text{Isin\_lut}_{(\phi t_n)}\cdot\text{Fcos\_lut}_{(\phi f_n)}, q2\right] + \text{QR}\left[\text{Icos\_lut}_{(\phi t_n)}\cdot\text{Fsin\_lut}_{(\phi f_n)}, q2\right]\right] \\
&\qquad \text{sin\_trig}_n \leftarrow \text{QR}\left(\text{sin\_trig}_n, q1\right) \\
&\quad \text{trig\_error} \leftarrow (\text{sin\_trig} - \text{sin\_ref}) \\
&\quad \text{SNR} \leftarrow 20\cdot\log\left(\frac{\text{stdev}(\text{sin\_ref})}{\text{stdev}(\text{trig\_error}) + \varepsilon}\right) \\
&\quad \text{result} \leftarrow \text{round}(\text{SNR})
\end{aligned}
$$

Figure (B-3): The Mathcad function `SNR_trig_R`.

$$
\begin{aligned}
&\text{errspec\_trig\_R}(M, I, \psi, \text{bits}, Ns, R) := \\
&\quad \text{result} \leftarrow 0 \\
&\quad \text{win} \leftarrow \text{hamming}(Ns) \\
&\quad \gamma \leftarrow 0.54 \\
&\quad q1 \leftarrow \frac{1}{2^{\text{bits}-1}} \\
&\quad q2 \leftarrow \frac{1}{2^{(2\cdot\text{bits})-1}} \\
&\quad \varepsilon \leftarrow \frac{1}{\infty} \\
&\quad \text{for } n \in 0..Ns-1 \\
&\qquad \text{sin\_ref}_n \leftarrow \sin\left[2\cdot\pi\cdot\frac{\text{mod}\left[(n\cdot\psi), 2^M\right]}{2^M}\right] \\
&\quad \text{for } n \in 0..2^I-1 \\
&\qquad \text{Isin\_lut}_n \leftarrow \text{QR}\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^I}\right), q1\right) \\
&\qquad \text{Icos\_lut}_n \leftarrow \text{QR}\left(\cos\left(2\cdot\pi\cdot\frac{n}{2^I}\right), q1\right) \\
&\quad \text{for } n \in 0..2^{M-I}-1 \\
&\qquad \text{Fsin\_lut}_n \leftarrow \text{QR}\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^M}\right), q1\right) \\
&\qquad \text{Fcos\_lut}_n \leftarrow \text{QR}\left(\cos\left(2\cdot\pi\cdot\frac{n}{2^M}\right), q1\right) \\
&\quad \text{for } n \in 0..Ns-1 \\
&\qquad \phi t_n \leftarrow \text{floor}\left[\frac{\text{mod}\left[(n\cdot\psi), 2^M\right]}{2^{M-I}}\right] \\
&\qquad \phi f_n \leftarrow \text{mod}\left[(n\cdot\psi), 2^M\right] - \left(2^{M-I}\right)\cdot\phi t_n \\
&\qquad \phi f_n \leftarrow 2^R\cdot\text{mod}\left(\text{floor}\left(\frac{\phi f_n}{2^R}\right), 2^{M-I-R}\right) \\
&\qquad \text{sin\_trig}_n \leftarrow \left[\text{QR}\left[\text{Isin\_lut}\left(\phi t_n\right)\cdot\text{Fcos\_lut}\left(\phi f_n\right), q2\right] + \text{QR}\left[\text{Icos\_lut}\left(\phi t_n\right)\cdot\text{Fsin\_lut}\left(\phi f_n\right), q2\right]\right] \\
&\qquad \text{sin\_trig}_n \leftarrow \text{QR}\left(\text{sin\_trig}_n, q1\right) \\
&\quad \text{trig\_error} \leftarrow (\text{sin\_trig} - \text{sin\_ref}) \\
&\quad \text{error\_spec} \leftarrow \text{mag}\left(\text{CFFT}\left(\overrightarrow{(\text{win}\cdot\text{trig\_error})}\right) + \varepsilon\right) \\
&\quad \text{for } n \in 0..Ns-1 \\
&\qquad \text{error\_spec}_n \leftarrow 20\cdot\log\left(2\cdot\frac{1}{\gamma}\cdot\text{error\_spec}_n\right) \\
&\quad \text{result} \leftarrow \text{error\_spec}
\end{aligned}
$$

Figure (B-4): The Mathcad function `errspec_trig_R`.

SNR_interp_HAS(N,M,I,ψ,bits,Ns,Nh,r1,r2,b)

result ← 0

F ← M − I

$q1 \leftarrow \frac{1}{2^{bits-1}}$

$q2 \leftarrow \frac{1}{2^{(2\cdot bits)-1}}$

$\varepsilon \leftarrow \frac{1}{\infty}$

for $n \in 0..2^{I}-1$

$Isin\_lut_n \leftarrow QR\left(\sin\left(2\pi\cdot\frac{n}{2^{I}}\right),q1\right)$

$Icos\_lut_n \leftarrow QR\left(\cos\left(2\pi\cdot\frac{n}{2^{I}}\right),q1\right)$

for $n \in 0..2^{F}-1$

$Fsin\_lut_n \leftarrow QR\left(\sin\left(2\pi\cdot\frac{n}{2^{M}}\right),q1\right)$

$Fcos\_lut_n \leftarrow QR\left(\cos\left(2\pi\cdot\frac{n}{2^{M}}\right),q1\right)$

for $h \in 1..Nh$

$a_h \leftarrow A(h,r1,r2,b)$

for $n \in 0..Ns-1$

$$wav\_ref_n \leftarrow \sum_{k=1}^{Nh}\left[a_k\cdot\sin\left[2\cdot k\cdot\pi\cdot\frac{mod\left[(n\cdot\psi),2^{M}\right]}{2^{M}}\right]\right]$$

$$wav\_ref \leftarrow \frac{wav\_ref}{max(wav\_ref)}$$

for $n \in 0..2^{I}-1$

$$sin\_lut_n \leftarrow QR\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^{I}}\right),q1\right)$$

for $n \in 0..Ns-1$

$\phi_n \leftarrow mod\left[(n\cdot\psi),2^{M}\right]$

$$wav\_interp_n \leftarrow \sum_{h=1}^{Nh}\left[a_h\cdot\sum_{k=0}^{N}\left[sin\_lut_{modn\left[\left(\mu\_I(\phi_n,M,F,h)-floor\left(\frac{N-1}{2}\right)+k\right),2^{I}\right]}\cdot QR\left[\prod_{j=0}^{N}if\left[j=k,1,\left(\frac{\mu\_\alpha(\phi_n,M,F,h)+floor\left(\frac{N-1}{2}\right)-j}{k-j}\right)\right],q2\right]\right]\right] \quad \text{if } N\geq 0$$

$$wav\_interp_n \leftarrow \sum_{k=1}^{Nh}\left[a_k\cdot\left(\begin{array}{l}QR\left(Isin\_lut_{mod\left(\mu\_I(\phi_n,M,F,k),2^{I}\right)}\cdot Fcos\_lut_{mod\left(\mu\_F(\phi_n,M,F,k),2^{F}\right)},q2\right)\ldots\\ +QR\left(Icos\_lut_{mod\left(\mu\_I(\phi_n,M,F,k),2^{I}\right)}\cdot Fsin\_lut_{mod\left(\mu\_F(\phi_n,M,F,k),2^{F}\right)},q2\right)\end{array}\right)\right] \quad \text{if } N<0$$

$wav\_interp_n \leftarrow QR(wav\_interp_n,q1)$

$$wav\_interp \leftarrow \frac{wav\_interp}{max(wav\_interp)}$$

interp_error ← (wav_interp − wav_ref)

$$SNR \leftarrow 20\cdot\log\left(\frac{stdev(wav\_ref)}{stdev(interp\_error)+\varepsilon}\right)$$

result ← round(SNR)

Figure (B-5): The Mathcad function `SNR_interp_HAS`.

errspec_interp_HAS (N,M,I,ψ,bits,Ns,Nh,r1,r2,b)

result ← 0

win ← hamming(Ns)

γ ← 0.54

F ← M − I

$q1 \leftarrow \frac{1}{2^{bits-1}}$

$q2 \leftarrow \frac{1}{2^{(2\cdot bits)-1}}$

$\varepsilon \leftarrow \frac{1}{\infty}$

for $n \in 0..2^{I}-1$

$Isin\_lut_n \leftarrow QR\left(\sin\left(2\pi\cdot\frac{n}{2^{I}}\right),q1\right)$

$Icos\_lut_n \leftarrow QR\left(\cos\left(2\pi\cdot\frac{n}{2^{I}}\right),q1\right)$

for $n \in 0..2^{F}-1$

$Fsin\_lut_n \leftarrow QR\left(\sin\left(2\pi\cdot\frac{n}{2^{M}}\right),q1\right)$

$Fcos\_lut_n \leftarrow QR\left(\cos\left(2\pi\cdot\frac{n}{2^{M}}\right),q1\right)$

for $h \in 1..Nh$

$a_h \leftarrow A(h,r1,r2,b)$

for $n \in 0..Ns-1$

$$wav\_ref_n \leftarrow \sum_{k=1}^{Nh}\left[a_k\cdot\sin\left[2\cdot k\cdot\pi\cdot\frac{mod\left[(n\cdot\psi),2^{M}\right]}{2^{M}}\right]\right]$$

$$wav\_ref \leftarrow \frac{wav\_ref}{max(wav\_ref)}$$

for $n \in 0..2^{I}-1$

$$sin\_lut_n \leftarrow QR\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^{I}}\right),q1\right)$$

for $n \in 0..Ns-1$

$\phi_n \leftarrow mod\left[(n\cdot\psi),2^{M}\right]$

$$wav\_interp_n \leftarrow \sum_{h=1}^{Nh}\left[a_h\cdot\sum_{k=0}^{N}\left[sin\_lut_{modn\left[\left(\mu\_I(\phi_n,M,F,h)-floor\left(\frac{N-1}{2}\right)+k\right),2^{I}\right]}\cdot QR\left[\prod_{j=0}^{N}if\left[j=k,1,\left(\frac{\mu\_\alpha(\phi_n,M,F,h)+floor\left(\frac{N-1}{2}\right)-j}{k-j}\right)\right],q2\right]\right]\right] \text{ if } N\geq 0$$

$$wav\_interp_n \leftarrow \sum_{k=1}^{Nh}\left[a_k\cdot\left(QR\left(Isin\_lut_{mod\left(\mu\_I(\phi_n,M,F,k),2^{I}\right)}\cdot Fcos\_lut_{mod\left(\mu\_F(\phi_n,M,F,k),2^{F}\right)},q2\right)\ldots\right.\right.$$
$$\left.\left.+QR\left(Icos\_lut_{mod\left(\mu\_I(\phi_n,M,F,k),2^{I}\right)}\cdot Fsin\_lut_{mod\left(\mu\_F(\phi_n,M,F,k),2^{F}\right)},q2\right)\right)\right] \text{ if } N<0$$

$wav\_interp_n \leftarrow QR\left(wav\_interp_n,q1\right)$

$$wav\_interp \leftarrow \frac{wav\_interp}{max(wav\_interp)}$$

interp_error ← (wav_interp − wav_ref)

$error\_spec \leftarrow mag\left(CFFT\left(\overrightarrow{(win\cdot interp\_error)}\right)+\varepsilon\right)$

for $n \in 0..Ns-1$

$$error\_spec_n \leftarrow 20\cdot\log\left(2\cdot\frac{1}{\gamma}\cdot error\_spec_n\right)$$

result ← error_spec

Figure (B-6): The Mathcad function `errspec_interp_HAS`.

$$\text{SNR\_interp\_PAS}(fs, B, N, M, I, \rho, bits, Ns, Np, r1, r2, b, v)$$

result ← 0

F ← M − I

$$\gamma \leftarrow 2^{\frac{1}{(12)\cdot 128}}$$

$$q1 \leftarrow \frac{1}{2^{bits-1}}$$

$$q2 \leftarrow \frac{1}{2^{(2\cdot bits)-1}}$$

$$\varepsilon \leftarrow \frac{1}{\infty}$$

for $n \in 0..2^{14}-1$

$$\Psi_n \leftarrow \left(\frac{B\cdot\gamma^n\cdot 2^M}{fs}\right)$$

for $h \in 1..Np$

$$a_h \leftarrow A(h, r1, r2, b)$$

for $h \in 1..Np$

$$k_h \leftarrow h$$

$$\text{if}(h \geq 2, \beta_h \leftarrow \text{round}(\text{rnd}(v)), \beta_h \leftarrow 0)$$

for $n \in 0..Ns-1$

$$wav\_ref_n \leftarrow \sum_{j=1}^{Np}\left[a_j\cdot\sin\left[2\cdot\pi\cdot\frac{\text{mod}\left[n\cdot\left[\frac{B\cdot\gamma^{\rho}\cdot 2^M}{fs}\cdot\left(\gamma^{\beta_j}+k_j-1\right)\right],2^M\right]}{2^M}\right]\right]$$

$$wav\_ref \leftarrow \frac{wav\_ref}{\max(\text{mag}(wav\_ref))}$$

for $n \in 0..2^I-1$

$$sin\_lut_n \leftarrow QR\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^I}\right), q1\right)$$

for $n \in 0..Ns-1$

$$\phi 1_n \leftarrow \text{mod}\left[\left(n\cdot\Psi_{\rho}\right),2^M\right]$$

for $h \in 1..Np$

$$\phi 2_h \leftarrow \text{mod}\left[\left[n\cdot\Psi_{(\rho+\beta_h)}\right],2^M\right]$$

$$\phi 3_h \leftarrow \text{mod}\left(\phi 2_h - \phi 1_n, 2^M\right)$$

$$\phi 4_h \leftarrow \text{mod}\left(k_h\cdot\phi 1_n, 2^M\right)$$

$$\phi 5_h \leftarrow \text{mod}\left(\phi 3_h + \phi 4_h, 2^M\right)$$

$$\phi t_h \leftarrow \text{floor}\left(\frac{\phi 5_h}{2^F}\right)$$

$$\alpha_h \leftarrow \left(\frac{\phi 5_h - 2^F\cdot\text{floor}\left(\frac{\phi 5_h}{2^F}\right)}{2^F}\right)$$

$$wav\_interp_n \leftarrow \sum_{h=1}^{Np}\left[a_h\cdot\sum_{m=0}^{N}\left[sin\_lut_{\text{modn}\left[\left(\phi t_h-\text{floor}\left(\frac{N-1}{2}\right)+m\right),2^I\right]}\cdot QR\left[\prod_{j=0}^{N}\text{if}\left[j=m,1,\left(\frac{\alpha_h+\text{floor}\left(\frac{N-1}{2}\right)-j}{m-j}\right)\right],q2\right]\right]\right]$$

$$wav\_interp_n \leftarrow QR\left(wav\_interp_n, q1\right)$$

$$wav\_interp \leftarrow \frac{wav\_interp}{\max(\text{mag}(wav\_interp))}$$

$$interp\_error \leftarrow (wav\_interp - wav\_ref)$$

$$SNR \leftarrow 20\cdot\log\left(\frac{\text{stdev}(wav\_ref)}{\text{stdev}(interp\_error)+\varepsilon}\right)$$

result ← (SNR)

Figure (B-7): The Mathcad function `SNR_interp_PAS`.

spec_interp_PAS_pitch (fs, B, N, M, I, ρ, bits, Ns, Np, r1, r2, b, β)

result ← 0

F ← M − I

win ← hamming(Ns)

$\gamma \leftarrow 2^{\frac{1}{(12)\cdot 128}}$

$q1 \leftarrow \frac{1}{2^{bits-1}}$

$q2 \leftarrow \frac{1}{2^{(2\cdot bits)-1}}$

$\varepsilon \leftarrow \frac{1}{\infty}$

for $n \in 0..2^{14}-1$

$\Psi_n \leftarrow \left(\frac{B\cdot\gamma^{n}\cdot 2^{M}}{fs}\right)$

for $h \in 1..Np$

$a_h \leftarrow A(h, r1, r2, b)$

for $h \in 1..Np$

$k_h \leftarrow h$

for $n \in 0..2^{I}-1$

$sin\_lut_n \leftarrow QR\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^{I}}\right), q1\right)$

for $n \in 0..Ns-1$

$\phi 1_n \leftarrow mod\left[\left(n\cdot\Psi_\rho\right), 2^{M}\right]$

for $h \in 1..Np$

$\phi 2_h \leftarrow mod\left[\left[n\cdot\Psi\left(\rho+\beta_h\right)\right], 2^{M}\right]$

$\phi 3_h \leftarrow mod\left(\phi 2_h - \phi 1_n, 2^{M}\right)$

$\phi 4_h \leftarrow mod\left(k_h\cdot\phi 1_n, 2^{M}\right)$

$\phi 5_h \leftarrow mod\left(\phi 3_h + \phi 4_h, 2^{M}\right)$

$\phi t_h \leftarrow floor\left(\frac{\phi 5_h}{2^{F}}\right)$

$\alpha_h \leftarrow \left(\frac{\phi 5_h - 2^{F}\cdot floor\left(\frac{\phi 5_h}{2^{F}}\right)}{2^{F}}\right)$

$wav\_interp_n \leftarrow \sum_{h=1}^{Np}\left[a_h\cdot\sum_{m=0}^{N}\left[sin\_lut_{modn\left[\left(\phi t_h - floor\left(\frac{N-1}{2}\right)+m\right), 2^{I}\right]}\cdot QR\left[\prod_{j=0}^{N} if\left[j=m, 1, \left(\frac{\alpha_h + floor\left(\frac{N-1}{2}\right) - j}{m-j}\right)\right], q2\right]\right]\right]$

$wav\_interp_n \leftarrow QR\left(wav\_interp_n, q1\right)$

$spec \leftarrow mag\left(CFFT\left(\overrightarrow{(win\cdot wav\_interp)}\right) + \varepsilon\right)$

for $n \in 0..Ns-1$

$spec_n \leftarrow 20\cdot\log\left(2\cdot\frac{1}{0.54}\cdot spec_n\right)$

result ← spec

result

Figure (B-8): The Mathcad function `spec_interp_PAS_pitch`.

spec_interp_PAS_freq(fs, B, N, M, I, ρ, bits, Ns, Np, r1, r2, b, β)

result ← 0

F ← M − I

win ← hamming(Ns)

$\psi \leftarrow \frac{2^{M} \cdot \beta}{fs}$

$\gamma \leftarrow 2^{\frac{1}{(12)\cdot 128}}$

$q1 \leftarrow \frac{1}{2^{bits-1}}$

$q2 \leftarrow \frac{1}{2^{(2\cdot bits)-1}}$

$\varepsilon \leftarrow \frac{1}{\infty}$

for $n \in 0..2^{14}-1$

$\Psi_n \leftarrow \left(\frac{B\cdot\gamma^{n}\cdot 2^{M}}{fs}\right)$

for $h \in 1..Np$

$a_h \leftarrow A(h, r1, r2, b)$

for $h \in 1..Np$

$k_h \leftarrow h$

for $n \in 0..2^{I}-1$

$sin\_lut_n \leftarrow QR\left(\sin\left(2\cdot\pi\cdot\frac{n}{2^{I}}\right), q1\right)$

for $n \in 0..Ns-1$

$\phi 1_n \leftarrow mod\left[\left(n\cdot\Psi_\rho\right), 2^{M}\right]$

for $h \in 1..Np$

$\phi 2_h \leftarrow mod\left[\left[n\cdot\left(\Psi_\rho + \psi_h\right)\right], 2^{M}\right]$

$\phi 3_h \leftarrow mod\left(\phi 2_h - \phi 1_n, 2^{M}\right)$

$\phi 4_h \leftarrow mod\left(k_h\cdot\phi 1_n, 2^{M}\right)$

$\phi 5_h \leftarrow mod\left(\phi 3_h + \phi 4_h, 2^{M}\right)$

$\phi t_h \leftarrow floor\left(\frac{\phi 5_h}{2^{F}}\right)$

$\alpha_h \leftarrow \left(\frac{\phi 5_h - 2^{F}\cdot floor\left(\frac{\phi 5_h}{2^{F}}\right)}{2^{F}}\right)$

$wav\_interp_n \leftarrow \sum_{h=1}^{Np}\left[a_h\cdot\sum_{m=0}^{N}\left[sin\_lut_{modn\left[\left(\phi t_h - floor\left(\frac{N-1}{2}\right)+m\right), 2^{I}\right]}\cdot QR\left[\prod_{j=0}^{N}\left[if\left[j=m, 1, \left(\frac{\alpha_h + floor\left(\frac{N-1}{2}\right) - j}{m-j}\right)\right], q2\right]\right]\right]$

$wav\_interp_n \leftarrow QR\left(wav\_interp_n, q1\right)$

$spec \leftarrow mag\left(CFFT\left(\overrightarrow{(win\cdot wav\_interp)}\right)\right) + \varepsilon$

for $n \in 0..Ns-1$

$spec_n \leftarrow 20\cdot\log\left(2\cdot\frac{1}{0.54}\cdot spec_n\right)$

result ← spec

result

Figure (B-9): The Mathcad function `spec_interp_PAS_freq`.

Appendix C The Order-3 Consecutive Access Vector Memory

This appendix considers the order-3 CAVM architecture which requires three distinct memory blocks, $\mathbf{B}_0$, $\mathbf{B}_1$ and $\mathbf{B}_2$, generating the data-parallel sample set $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1], \mathbf{T}[\phi_I + 2]\}$ needed for a quadratic interpolation of the wavetable vector $\mathbf{T}$ (i.e. $N = 2$ and $k = 3$). Samples are allocated to individual memory blocks from the wavetable vector, $\mathbf{T}$, in increments of three as illustrated in Figure (C-4) for the example case when $L = 9$. We also have three sample-type indices, p_0, p_1 and p_2, which take on values from the set $\{0, 1, 2\}$.

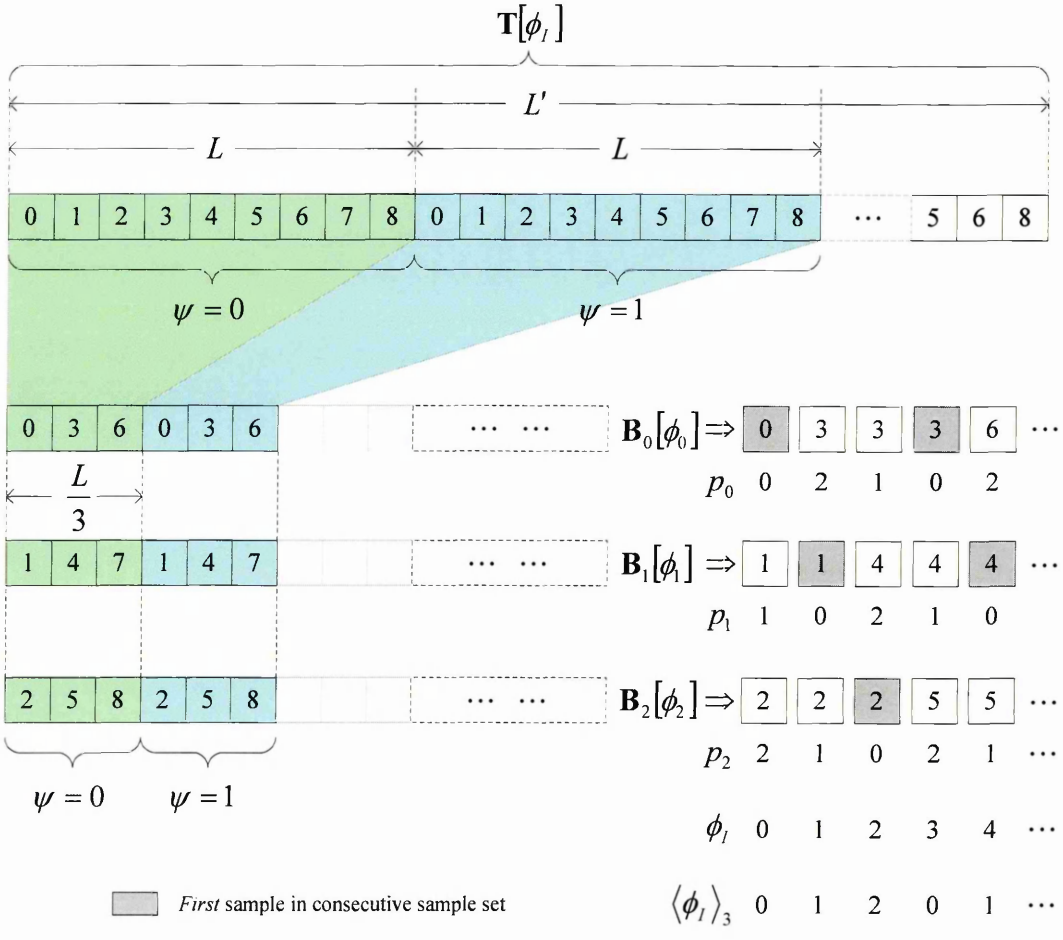


Figure (C-4): Memory allocation for the order-3 CAVM with $L = 9$.

Memory blocks $\mathbf{B}_0$, $\mathbf{B}_1$ and $\mathbf{B}_2$ are therefore written with data from $\mathbf{T}$ according to:

$$\mathbf{B}_0[m] = \mathbf{T}[n] \quad \text{for} \quad n = 3m$$

$$\mathbf{B}_1[m] = \mathbf{T}[n] \quad \text{for} \quad n = 3m+1 \qquad \text{(C-3)}$$

$$\mathbf{B}_2[m] = \mathbf{T}[n] \quad \text{for} \quad n = 3m+2$$

$$m \in [0, \frac{L'}{3} - 1], \quad \frac{L}{3} \in Z$$

The phase index, ϕ_I, is partitioned into three physical block addresses, $\phi_i \in [0, \frac{L}{3} - 1]$, with $i \in [0,2]$ which respectively address memory blocks $\mathbf{B}_i$ modulo-$\frac{L}{3}$ and generate the sample-set $\{\mathbf{T}[\phi_I], \mathbf{T}[\phi_I + 1], \mathbf{T}[\phi_I + 2]\}$ for $\phi_I \in [0, 2^I - 1]$.

Since we have three memory blocks, there is no longer a simple allocation of samples and corresponding definition of sample-type indices, p_0, p_1 and p_2 according to whether ϕ_I is odd or even. Instead, the block addresses are obtained from ϕ_I through modular division by 3. ϕ_2 is obtained by taking the integer part of $\frac{\phi_I}{3}$, modulo-$\frac{L}{3}$ (i.e. $\left\langle \left\lfloor \frac{\phi_I}{3} \right\rfloor \right\rangle_{\frac{L}{3}}$), with ϕ_1 and ϕ_0 obtained by offsetting ϕ_I by 1 and 2, respectively, before the modular division by 3. Hence, for the order-3 CAVM and L exactly divisible by 3, these mappings are defined by the expressions:

$$\phi_0 = \left\langle \left\lfloor \frac{\phi_I + 2}{3} \right\rfloor \right\rangle_{\frac{L}{3}} \qquad \phi_1 = \left\langle \left\lfloor \frac{\phi_I + 1}{3} \right\rfloor \right\rangle_{\frac{L}{3}} \qquad \phi_2 = \left\langle \left\lfloor \frac{\phi_I}{3} \right\rfloor \right\rangle_{\frac{L}{3}} \qquad \text{(C-4)}$$

Table (C-2) illustrates the block address and sample-type index sequences for an order-3 CAVM with $L = 9$.

ϕ_I	ϕ_0	ϕ_1	ϕ_2	$\mathbf{B}_0[\phi_0]$	p_0	$\mathbf{B}_1[\phi_1]$	p_1	$\mathbf{B}_2[\phi_2]$	p_2	**Sample Set**
0	0	0	0	T[0]	0	T[1]	1	T[2]	2	{T[0], T[1], T[2]}
1	1	0	0	T[3]	2	T[1]	0	T[2]	1	{T[1], T[2], T[3]}
2	1	1	0	T[3]	1	T[4]	2	T[2]	0	{T[2], T[3], T[4]}
3	1	1	1	T[3]	0	T[4]	1	T[5]	2	{T[3], T[4], T[5]}
4	2	1	1	T[6]	2	T[4]	0	T[5]	1	{T[4], T[5], T[6]}
5	2	2	1	T[6]	1	T[7]	2	T[5]	0	{T[5], T[6], T[8]}
6	2	2	2	T[6]	0	T[7]	1	T[8]	2	{T[6], T[7], T[8]}
7	0	2	2	T[0]	2	T[7]	0	T[8]	1	{T[7], T[8], T[0]}
8	0	0	2	T[0]	1	T[1]	2	T[8]	0	{T[8], T[0], T[1]}

Table (C-2): Order-3 CAVM address sequences, memory block data values and sample-type indices for $L = 9$. The $\mathbf{B}_0[\phi_0]$, $\mathbf{B}_1[\phi_1]$ and $\mathbf{B}_2[\phi_2]$ columns illustrate sample ordering permutations at the memory block outputs.

For $\phi_I \in [0, 2^I - 1]$, the order-3 sample-type indices take values that follow a cyclic permutation of the set $\{0, 1, 2\}$ obtained from the modulo operation $\langle \phi_I \rangle_3 = \phi_I - 3\left\lfloor \frac{\phi_I}{3} \right\rfloor \in \{0, 1, 2\}$ for all $\phi_I \in [0, 2^I - 1]$. The permuted sets and hence the sample-type indices are obtained by offsetting ϕ_I prior to the modulo-3 operation and then subtracting the result from 2. The sample-type indices for the order-3 CAVM are therefore given by:

$$p_0 = 2 - \langle \phi_I + 2 \rangle_3 \qquad p_1 = 2 - \langle \phi_I + 1 \rangle_3 \qquad p_2 = 2 - \langle \phi_I \rangle_3 \qquad \text{(C-5)}$$

Figure (C-5) illustrates the arithmetic processing model for an order-3 CAVM and linear interpolation processor assuming a fractional phase address, $(\phi_I + \alpha)$ and wavetable index, ψ. This model requires three interpolation coefficients which are dependent on the fractional address, α, and the three sample-type indices, p_0, p_1 and p_2, to reorder the memory outputs.

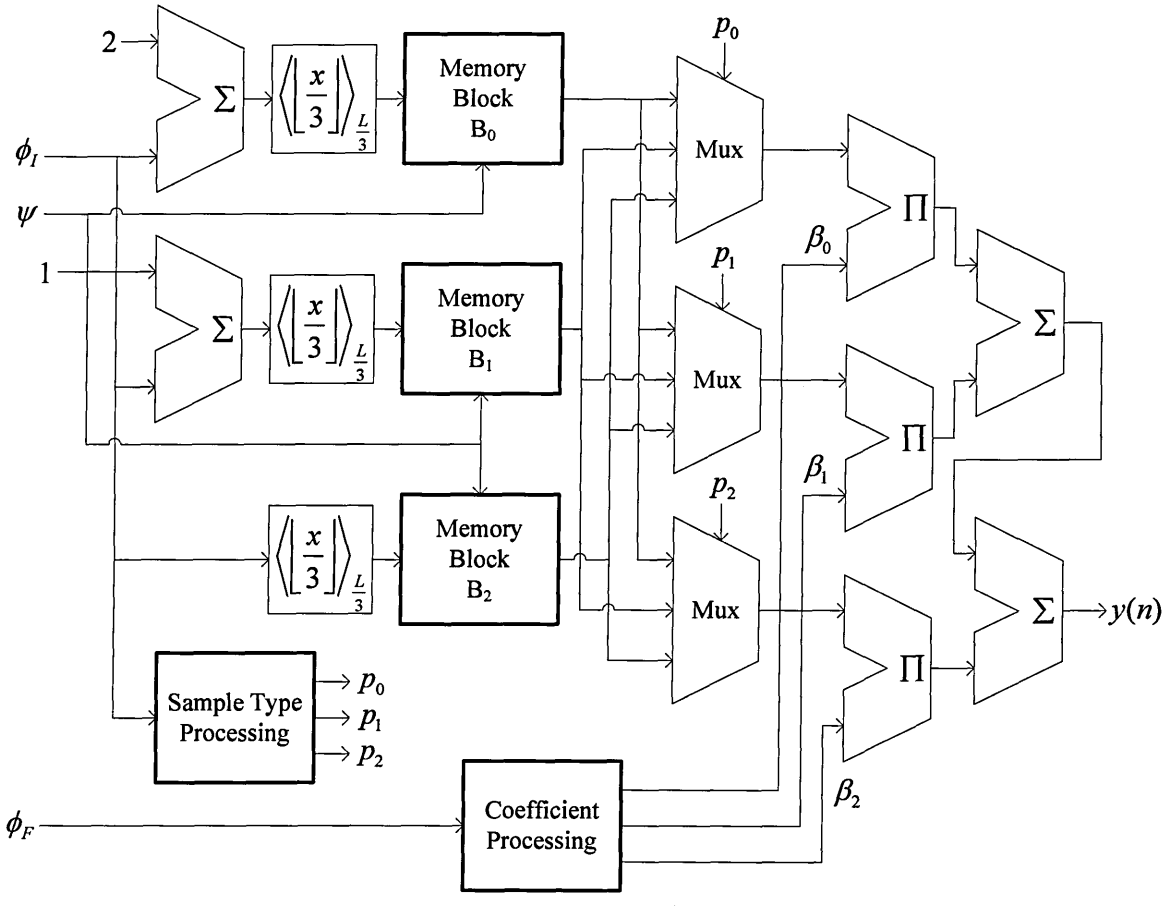


Figure (C-5): Order-3 CAVM and quadratic interpolation processing model.

The ϕ_i and p_i terms with $i \in [0, k-1]$ are dependent on a modular division-by-3 operation which is not effectible with a single shift operation. Parhami [2000] reports that modular division by constant integer dividends can be reduced to a sequence of shift and add operations by using the mathematical property that for every odd integer,

a, there exists an odd integer, b, such that $ab = 2^n - 1$, where n is a positive integer. Hence, we have:

$$\frac{1}{a} = \frac{b}{2^n - 1} = \frac{b}{2^n\left(1 - 2^{-n}\right)}$$

$$= \frac{b}{2^n}\left(1 + 2^{-n}\right)\left(1 + 2^{-2n}\right)\left(1 + 2^{-4n}\right)\ldots$$

$$= \frac{b}{2^n}\prod_{i=0}^{P}\left(1 + 2^{-2^i n}\right) \qquad \text{(C-6)}$$

$$i \in [0, P]$$

where P denotes the number of product terms required to achieve a particular quotient accuracy and is proportional to the logarithm of the dividend word width in bits [Parhami, 2000]. Eq. (C-6) indicates that to divide x by a, we first multiply x by $\frac{b}{2^n}$, which is computable by a combination of left and right shift operations followed by P factors of the form $\left(1 + 2^{-j}\right)$, where $j = n, 2n, 4n, \ldots, 2^P n$. Each of these factors is computed by a right shift followed by an addition. Since we are concerned with division of x by 3, we have $a = 3$, $b = 5$ and $n = 4$ which gives:

$$\frac{x}{3} = \frac{5x}{16}\left(1 + 2^{-4}\right)\left(1 + 2^{-8}\right)\left(1 + 2^{-16}\right)\ldots \qquad \text{(C-7)}$$

In general, the number of $\left(1 + 2^{-j}\right)$ factors is bound by the number of bits required to address a particular CAVM memory block. For a 12 bit x value (i.e. CAVM address), Eq. (C-7) reduces to $\frac{x}{3} \approx \frac{5x}{16}\left(1 + 2^{-4}\right)\left(1 + 2^{-8}\right)$ since subsequent product terms do not contribute any further precision to a 12 bit result.

Eq. (C-6) yields an approximation to the quotient which is always *lower* than the true result necessitating care when truncating the fraction field to effect the correct modulo

division. For example, $\frac{4095}{3}$ is *precisely* 1365 but applying Eq. (C-6) estimates this quotient as ≈1364.979... yielding the incorrect modular division result of 1364 when the fraction field is truncated. This error is prevented for *positive dividends* if we increment the dividend by 1 prior to applying Eq. (C-6).

We now define an algorithm for modular division by 3 to 12 bit precision comprising only shift and add operations, thus:

$$
\begin{array}{ll}
q \leftarrow x+1 & \text{compute } x+1 \\
q \leftarrow q+\overleftarrow{2}(q) & \text{compute } 5(x+1) \\
q \leftarrow q+\vec{4}(q) & \text{compute } 5(x+1)\left(1+2^{-4}\right) \\
q \leftarrow q+\vec{8}(q) & \text{compute } 5(x+1)\left(1+2^{-4}\right)\left(1+2^{-8}\right) \\
q \leftarrow \vec{4}(q) & \text{compute } \dfrac{5(x+1)\left(1+2^{-4}\right)\left(1+2^{-8}\right)}{16} \\
q \leftarrow \lfloor q \rfloor & \text{remove fraction field}
\end{array}
\qquad \text{(C-8)}
$$

where $\overleftarrow{S}(x)$ denotes an S bit *left* shift applied to x and $\vec{S}(x)$ denotes an S bit *right* shift applied to x. The algorithm defined by Eq. (C-8) requires three additions, four shift operations and a final truncation operation to effect modular division by 3 to 12 bit precision and may be implemented by the serial data-flow processing model illustrated in Figure (C-6), where the carry-in to the first adder is set to 1 to effect the dividend unit increment. Further precision is readily achieved by adding further shift and add $\left(1+2^{-j}\right)$ terms as required for larger address word sizes.

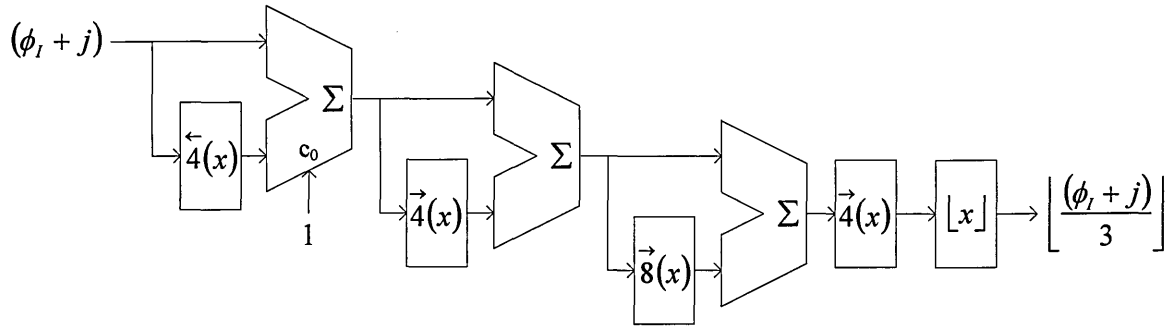


Figure (C-6): Arithmetic processing model for modular division-by-3 to 12-bit precision.